▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación

21

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informática y programación 21

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO.

Directora de producción:
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.

Diseño y maquetación:
BRAVO-LOFISH.

Fotografía:
EQUIPO GALATA.

Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de AULA DE INFORMATICA APLICADA (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomunicación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: (desde el tomo 5): Juan José Gómez, Licenciado en Química. LOGO: Cristina Manzanera, Licenciada en Informática. APLICACIONES: Sociedad Tamariz, Diplomada en Telecomunicación. OTROS LENGUAJES (COBOL): Eloy Pérez, Licenciado en Informática. Ana Pastor, Licenciada en Informática.

Ediciones Siglo Cultural, S.A.

Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.

Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.

Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-158-4
ISBN de la obra: 84-7688-068-7

Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M-5-677-1987
Printed in Spain - Impreso en España.

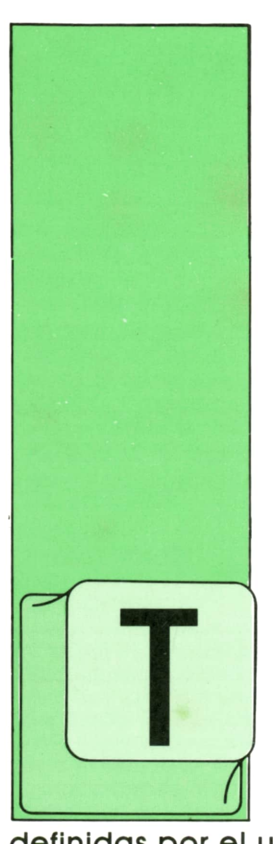
Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.

Agosto, 1987.
P.V.P. Canarias: 335,–.

INDICE

4 BASIC
9 MAQUINA Z-80
12 PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS
21 TECNICAS DE ANALISIS
23 TECNICAS DE PROGRAMACION
27 LOGO
31 PASCAL
35 OTROS LENGUAJES

BASIC



 FUNCIONES (I)

TODAS las versiones BASIC de los distintos ordenadores permiten el manejo de diversas funciones, tanto las funciones estándar suministradas como las funciones definidas por el usuario.

En este tomo vamos a comenzar por el estudio de las primeras, dejando las funciones definidas para tomos posteriores.

En primer lugar, debemos tener una idea del concepto de función. A grandes rasgos podríamos considerar una función como un pequeño programa dentro del programa que estemos desarrollando. Muchas veces es necesario hacer una operación determinada, como, por ejemplo, saber el número de caracteres que tiene una cadena o, simplemente, hacer la raíz cuadrada de un número. Estas y otras muchas operaciones se pueden realizar mediante el uso de las funciones que el BASIC pone a nuestra disposición.

Por regla general todas las funciones tienen el siguiente formato:

NOMBRE (argumento/s)

donde **NOMBRE** es el nombre específico de cada función. Entre paréntesis van uno o varios **argumentos** (separados por comas) dependiendo de la función.

Los argumentos pueden ser de carácter numérico o alfanumérico (constantes, variables o expresiones), según la función aplicada.

Por otra parte, siempre que aplicamos una función sobre su(s) argumento(s) correspondiente(s) se produce un resultado que también pueder ser numérico o alfanumérico. Atendiendo al tipo de resultado, podemos hacer la siguiente clasificación:

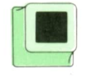
FUNCIONES NUMERICAS:

- — Argumento numérico.
- — Argumento alfanumérico.

FUNCIONES ALFANUMERICAS:

- — Argumento numérico.
- — Argumento alfanumérico.

Vamos a comenzar con el estudio detallado de las principales funciones numéricas con argumento numérico, dejando el estudio de los demás tipos para tomos sucesivos. Pero antes conviene advertir que, dependiendo de cada fabricante, cada ordenador puede disponer de más o menos funciones. Pretender estudiarlas todas aquí sería una tarea prácticamente imposible, por tanto, para conocer todas las funciones disponibles en cada ordenador podemos consultar el manual correspondiente.

 Funciones numéricas con argumento numérico

Valor absoluto: ABS

El formato general es el siguiente:

ABS(N)

Por tanto, acepta un solo argumento. El resultado es el valor absoluto de dicho argumento, es decir, su mismo valor, pero con signo positivo.

Veamos algunos ejemplos, así como el resultado que aparecería en pantalla:

Comando directo	Resultado
PRINT ABS(8*4−6/2)	29
PRINT ABS(3−6*4+5)	16
LET A=−69:PRINT ABS(A)	69

Conversión a entero: CINT

El formato general es el siguiente:

CINT(N)

Admite un único formato y devuelve un número entero como resultado del redondeo.

Veamos algunos ejemplos:

Comando directo	Resultado
PRINT CINT(18.725)	19
PRINT CINT(10/3−5*2)	−7
LET A=CINT(−28.3):PRINT A	−28

Parte entera: INT

El formato general es el siguiente:

INT(N)

Utiliza un solo argumento y devuelve el número entero de valor inferior más próximo.

Por ejemplo:

Comando directo	Resultado
PRINT INT(68.925)	68
PRINT INT(5/3−2*4)	−7
LET A=−2.8:PRINT INT(A)	−3

Sin decimales: FIX

El formato general es:

FIX(N)

Acepta un único argumento. Devuelve el valor entero del argumento despreciando los decimales y sin redondear.

Veamos algunos ejemplos:

Comando directo	Resultado
PRINT FIX(−18.9)	−18
PRINT FIX(20/3)	6
LET A=FIX(21/10+2):PRINT A	4

Redondeo: ROUND

El formato general es el siguiente:

ROUND(N,M)

Por tanto, admite dos argumentos numéricos. El primero, N, es un número real cualquiera, mientras que el segundo, M, indica el número de decimales que deseamos en el redondeo de N. Este segundo argumento se puede omitir, en cuyo caso ROUND funcionará exactamente igual que CINT.

Veamos algunos ejemplos:

Comando directo	Resultado
PRINT ROUND(8.2374,2)	8.24
PRINT ROUND(−2*3.14159,4)	−6.2832
LET A=5.82918:PRINT ROUND(A,3)	5.829

Signo: SGN

El formato es el siguiente:

SGN(N)

Acepta un solo argumento. El resultado es 1 si el argumento es positivo, 0 si es 0, y −1 si es negativo.

Por ejemplo:

Comando directo	Resultado
PRINT SGN(105.82)	1
PRINT SGN(6*4−8*3)	0
LET A=SGN(−7):PRINT A	−1

Raíz cuadrada: SQR

El formato es:

SQR(N)

Devuelve la raíz cuadrada del argumento que, lógicamente, debe ser positivo, ya que el ordenador no maneja números complejos.

Veamos algunos ejemplos:

Comando directo	Resultado
PRINT SQR(25)	5
PRINT SQR(4^2+3*8)	6.3245553
LET A=107.5:PRINT SQR(A)	10.36822

Exponencial: EXP

El formato general es el siguiente:

EXP(N)

Admite un argumento numérico de cualquier tipo y devuelve el resultado de elevar el número **e** al argumento indicado.

El número **e** es un número irracional, cuyo valor redondeado a cuatro decimales es 2.7182.

Veamos algunos ejemplos:

Comando directo	Resultado
PRINT EXP(1)	2.71828183
PRINT EXP(3−5)	0.13533528
LET A=EXP(0):PRINT A	1

Logaritmo: LOG

Esta función es la inversa de EXP. Su formato es el siguiente:

LOG(N)

El argumento es numérico, pero siempre debe ser positivo. El resultado es el logaritmo natural o neperiano, es decir, en base **e,** del argumento.

Por ejemplo:

Comando directo	Resultado
PRINT LOG(EXP(1))	1
PRINT LOG(8−2*3)	0.69314718
LET A=1:PRINT LOG(A)	0

Funciones trigonométricas

La mayoría de los ordenadores incorporan en sus versiones de BASIC las siguientes funciones trigonométricas:

Función	Formato
Seno	SIN(N)
Coseno	COS(N)
Tangente	TAN(N)
Arcotangente	ATN(N)
Arcoseno	ASN(N)
Arcocoseno	ACS(N)

El argumento indica el ángulo en radianes. Recordemos que:

1 RADIAN = 180/PI GRADOS SEXAGESIMALES

por tanto, si manejamos grados sexagesimales tendremos que hacer una conversión a radianes.

Muchos ordenadores disponen de la función **PI,** que devuelve el valor de este número irracional. Si nuestro ordenador no dispusiera de PI, tendríamos que introducir previamente su valor (3.141592...) en forma de constante, o bien asignándolo previamente a una variable numérica.

Memoria libre: FRE

El formato es el siguiente:

FRE(N)

Admite un argumento de cualquier tipo, incluso alfanumérico, que se denomina «argumento mudo», ya que, aunque es necesario, su valor no está relacionado con el resultado. Devuelve el espacio libre que queda en la memoria del ordenador.

Números aleatorios: RND y RANDOMIZE

El formato general es el siguiente:

RND(N)

aunque en algunos ordenadores no es necesario especificar argumento alguno.

La función RND tiene por objeto la generación de números aleatorios; por tanto, se emplea mucho en el diseño de programas de juegos donde interviene a menudo el azar.

En la mayoría de los ordenadores, como AMSTRAD, IBM o MSX, el efecto es distinto, dependiendo del signo del argumento.

— Cuando el argumento es positivo, devuelve un número aleatorio comprendido entre 0 y 1, pero teniendo en cuenta que el 0 está incluido en el intervalo de posibilidades, mientras que el 1 no.

— Cuando el argumento es cero devuelve el último aleatorio generado.

— Cuando el argumento es negativo devuelve el número aleatorio, que es siempre el mismo para cada valor del argumento. Además, se inicializa de nuevo en memoria la secuencia de números al azar.

En el Commodore la función RND devuelve unos resultados similares a los explicados, aunque en el caso de que el argumento sea 0 el efecto será el mismo que si fuera positivo.

De cualquier forma, normalmente se utilizan argumentos positivos.

En el caso del SPECTRUM la función RND no utiliza argumento de ningún tipo y su misión es generar un número al azar entre 0 y 1.

En muchos ordenadores la función RND se suele utilizar en combinación con RANDOMIZE, que permite elegir el punto en el que se inicia la secuencia de números aleatorios. Su formato general es el siguiente:

RANDOMIZE(N)

El efecto más aleatorio se consigue cuando el inicio de la secuencia está en función del tiempo que lleva encendido el ordenador. Esto se consigue con:

RANDOMIZE TIME en el AMSTRAD
RANDOMIZE TIMER en el IBM
RANDOMIZE 0 en el SPECTRUM

Los MSX no disponen de la función RANDOMIZE, sin embargo, podemos conseguir un efecto parecido utilizando para RND el siguiente formato:

RND(−TIME)

El COMMODORE no dispone de RANDOMIZE.

Veamos un ejemplo. El programa 1 simula el lanzamiento de un dado por el ordenador. Además, nos permite apostar a un número.

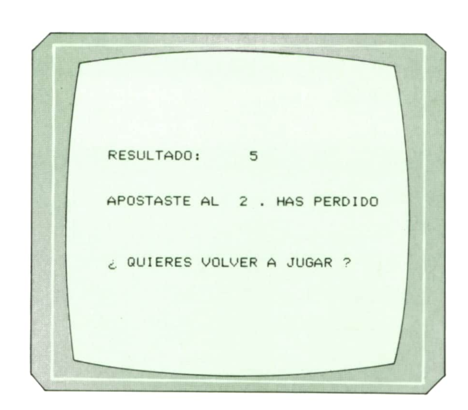
```
10 REM ****************
20 REM *     DADO     *
30 REM ****************
40 CLS
50 INPUT "¿ A QUE NUMERO APUESTAS ?";N
60 CLS
70 IF N<1 OR N>6 THEN GOTO 50
80 RANDOMIZE TIMER
90 LET DADO=INT(RND(1)*6)+1
100 PRINT "RESULTADO:",DADO
110 PRINT :PRINT
120 IF DADO=N THEN PRINT "¡ ACERTASTE !
    HAS GANADO":GOTO 140
130 PRINT "APOSTASTE AL ";N;". HAS PERDIDO"
140 PRINT :PRINT :PRINT
150 INPUT "¿ QUIERES VOLVER A JUGAR ?";R$
160 IF R$="S" OR R$="s" THEN GOTO 40
170 IF R$<>"N" AND R$<>"n" THEN GOTO 150
```

En la línea 90 se simula perfectamente el dado, puesto que se genera al azar un número entero (función INT) comprendido entre 1 y 6, ya que la función RND va multiplicada por 6 (el resultado estaría entre 0 y 5) y después se suma 1 al número obtenido.

Recordemos que en el SPECTRUM habría que suprimir el argumento de RND y que la función RANDOMIZE varía de unos ordenadores a otros (en algunos casos habrá que suprimirla).

En la figura 1 podemos ver el aspecto de la pantalla tras una ejecución.

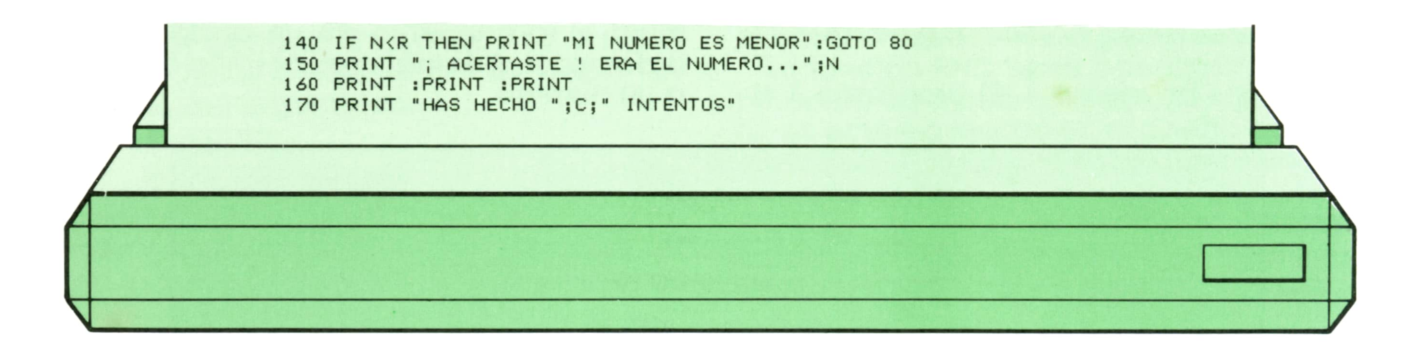
El programa 2 es un pequeño juego. El ordenador selecciona un número al azar entre 1 y 100 y el usuario tiene que acertarlo. Para ello el ordenador le ayuda con algunas pistas.





Presentación en pantalla del programa 1.

```
10 REM **********************
20 REM * ADIVINAR UN NUMERO *
30 REM **********************
40 CLS
50 RANDOMIZE TIMER
60 LET N=INT(RND(1)*100)+1
70 LET C=0
80 INPUT "DIME UN NUMERO ";R
90 CLS
100 LET C=C+1
110 IF R<1 OR R>100 THEN PRINT "MI NUMERO ESTA ENTRE 1 Y 100":GOTO 80
120 IF R<>INT(R) THEN PRINT "MI NUMERO ES ENTERO":GOTO 80
130 IF N>R THEN PRINT "MI NUMERO ES MAYOR":GOTO 80
```



En la línea 60 se genera el número aleatorio entre 1 y 100. Para ello se sigue el mismo proceso que en el programa anterior.

En la línea 70 se establece un contador para contabilizar el número de intentos hasta acertar el número.

La condición de la línea 120 sirve para advertirnos, cuando respondamos un número decimal, que el ordenador ha «pensado» un número entero.

Las demás condiciones están bastante claras. Todas tienen por objeto dar alguna pista sobre el número que hay que adivinar. Una vez que acertemos el número aparecerá una pantalla similar a la de la figura 2.




Presentación en pantalla del programa 2.

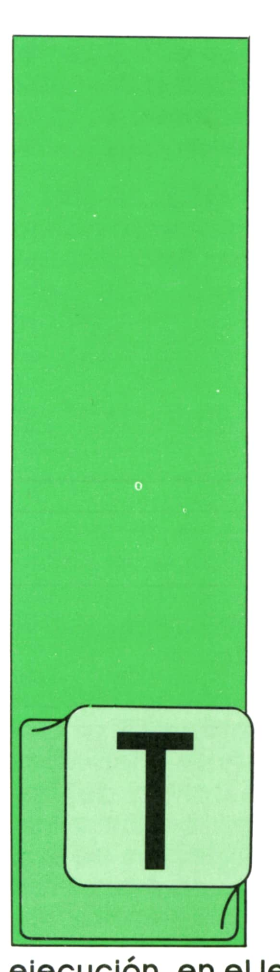
Para finalizar, en la figura 3 podemos ver una tabla con las principales funciones numéricas de argumento numérico disponibles en cada ordenador.

	AMSTRAD	COMMODORE	IBM	MSX	SPECTRUM
ABS	*	*	*	*	*
CINT	*		*	*	
INT	*	*	*	*	*
FIX	*		*	*	
ROUND	*				
SGN	*	*	*	*	*
SQR	*	*	*	*	*
EXP	*	*	*	*	*
LOG	*	*	*	*	*
SIN	*	*	*	*	*
COS	*	*	*	*	*
TAN	*	*	*	*	*
ATN	*	*	*	*	*
ASN					*
ACS					*
PI	*	*			
FRE	*	*	*	*	
RND	*	*	*	*	*
RANDOMIZE	*		*		*


Tabla-resumen de funciones numéricas (argumento numérico).

MAQUINA Z-80

SPECTRUM, AMSTRAD, MSX



Instrucciones de salto y otras de interés

RATAREMOS en este apartado, en primer lugar, de las instrucciones que se refieren al cambio en la secuencia normal de ejecución, en el lenguaje máquina del Z-80. Luego hablaremos de otras instrucciones de uso general.

En las instrucciones de salto podemos diferenciar dos tipos, las que se refieren a salto a subrutina, de las que ya tratamos al hablar del stack; y las de salto propiamente dichas. De estas últimas nos ocuparemos ahora. Para ello describiremos brevemente cada una de ellas (ver figura 1).

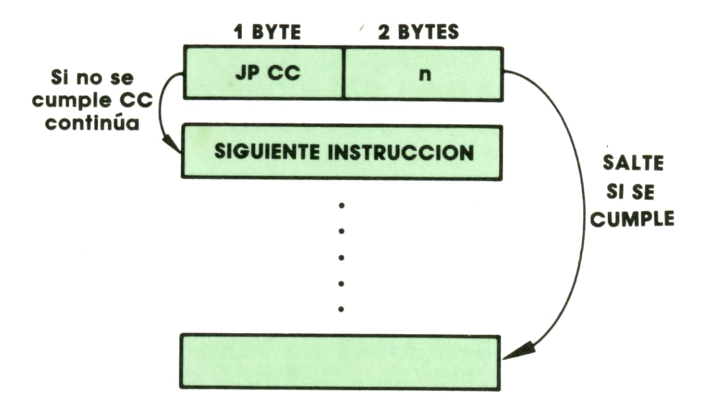
Grupo de salto

Código mnemotécnico	Operación simbólica	Indicadores S Z H P/V N C	Códigos 76 543 210 Hex	N.° de bytes	N.° de ciclos M	N.° de estados T	Comentarios
JP nn	PC←nn	• • X • X • • •	11 000 011 C3 ←n→ ←n→	3	3	10	
JP cc,nn	Si la condición cc es cierta, PC←nn; si no, se continúa	• • X • X • • •	11 cc 010 ←n→ ←n→	3	3	10	cc Condición 000 NZ no cero 001 Z cero 010 NC no arrastre 011 C arrastre 100 PO paridad impar 101 PE paridad par 110 P signo positivo 111 M signo negativo
JR e	PC←PC+e	• • X • X • • •	00 011 000 18 ←e−2→	2	3	12	
JR C,e	Si C=0, se continúa	• • X • X • • •	00 111 000 38 ←e−2→	2	2	7	Si no se cumple la condición
	Si C=1, PC←PC+e			2	3	12	Si se cumple la condición
JR NC,e	Si C=1, se continúa	• • X • X • • •	00 110 000 30 ←e−2→	2	2	7	Si no se cumple la condición
	Si C=0, PC←PC+e			2	3	12	Si se cumple la condición
JR Z,e	Si Z=0, se continúa	• • X • X • • •	00 101 000 28 ←e−2→	2	2	7	Si no se cumple la condición
	Si Z=1, PC←PC+e			2	3	12	Si se cumple la condición
JR NZ,e	Si Z=1, se continúa	• • X • X • • •	00 100 000 20 ←e−2→	2	2	7	Si no se cumple la condición
	Si Z=0, PC←PC+e			2	3	12	Si se cumple la condición
JP (HL)	PC←HL	• • X • X • • •	11 101 001 E9	1	1	4	
JP (IX)	PC←IX	• • X • X • • •	11 011 101 DD 11 101 001 E9	2	2	8	
JP (IY)	PC←IY	• • X • X • • •	11 111 101 FD 11 101 001 E9	2	2	8	
DJNZ, e	B←B−1 Si B=0, se continúa	• • X • X • • •	00 010 000 10 ←e−2→	2	2	8	Si B=0
	Si B≠0, PC←PC+e			2	3	13	Si B≠0

NOTAS: e es la magnitud del salto relativo; es un número con signo del intervalo −126...129. en el código figura e−2 para provocar un salto de magnitud e, ya que PC se incrementa en 2 antes de sumarle el desplazamiento.

— En primer lugar, tenemos la instrucción JPnn. Su función es directa. Simplemente produce un salto incondicional a la posición de memoria referida como nn. Por ejemplo: JP $A000, transferirá el control a la instrucción que comienza en la dirección hexadecimal A000.

— La instrucción del tipo JP cc,nn transferirá el control a nn si se cumple la condición cc.



CC puede ser cualquiera de los valores NZ, Z, NC, C, PO, P y M. Estas letras significan por el mismo orden las siguientes condiciones: «distinto a cero», «igual a cero», «el acarreo es nulo», «el acarreo no es nulo», «la paridad es impar», «la paridad es par», «el signo es positivo», y «el signo es negativo».

Por supuesto, todas estas condiciones se refieren al contenido del acumulador y coinciden con el estado de los flags (o banderas) del registro F.

— La instrucción **JR e,** produce un salto incondicional, como la instrucción JP, pero relativo, **e** posiciones. Esto quiere decir que la ejecución continuará **e** posiciones más allá que en la que se encuentra la instrucción JR. **e** es un número en el intervalo −126... 129 (entero, por supuesto).

— La instrucción **JRC,e** produce el mismo efecto que la anterior, pero sólo cuando se cumple la condición C, es decir, cuando el contenido del acarreo no es nulo. En caso contrario, cuando el acarreo es igual a cero, la instrucción se ignora, y la ejecución continúa en la siguiente instrucción.

— La instrucción **JR NC,e** tiene el mismo efecto que la anterior, pero con la condición inversa (carry = cero).

— Las instrucciones **JR Z,e y JR NZ,e** tienen un funcionamiento similar a las anteriores, pero con las condiciones Z (igual a cero) y NZ (distinto de cero).

— Las instrucciones **JP (HL), JP (IX) y JP (IY)** provocan un salto indirecto con la dirección entre paréntesis. Esto supone que la ejecución continuará en la posición de memoria cuya dirección se encuentre en los registros de 16 bits entre paréntesis.

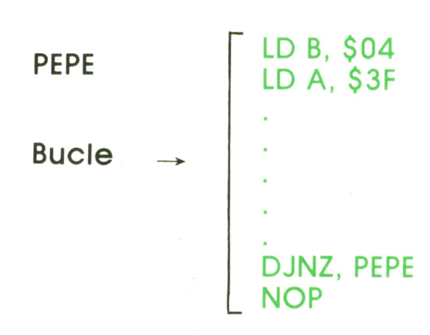
Por ejemplo, si el contenido de HL es $01FA, la instrucción JP (HL) será equivalente a **JP $01FA.**

— Por último, hablemos de la instrucción DJNZ,e. Esta instrucción está especialmente indicada para la realización de bucles en máquina, realizando varias acciones simultáneamente:

Cada vez que la instrucción sea ejecutada, el contenido del registro B se decrementa en 1 (lógicamente, debemos poner en B la variable de control del bucle). Una vez hecho esto, la misma instrucción comprueba el contenido de B; si éste es cero (fin del bucle), la ejecución continúa en la posición siguiente. En caso contrario (B<>0), se produce un salto relativo de **e** posiciones.

Como ya explicamos anteriormente, el salto relativo de **e** posiciones provoca que la ejecución continúe en la posición actual +e.

Veamos el uso de esta instrucción con un ejemplo.



1.° Cargamos en el registro B del número 4 (con lo que pretendemos que el bucle se repita cuatro veces).

2.° Se ejecuta el bucle, y al llegar a la instrucción DJNZ, ésta hace B = 3 (al decrementar) y provoca un salto relativo a la dirección PEPE.

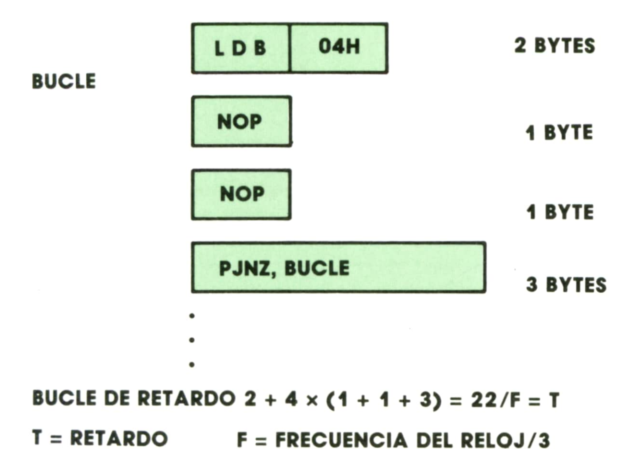
3.° El bucle se repetirá, y cuando se ejecute DJNZ y B = 0 (la ejecución continuará en la instrucción NOP).

Por último, describiremos las instrucciones varias. Son todas aquellas instrucciones que por su función específica no se pueden incluir en ninguno de los grupos anteriores (ver figura 3).

Grupo de salto
Grupo de instrucciones varias

Código mnemónico	Operación simbólica	Indicadores	Códigos	n.° de bytes	n.° de ciclos M	n.° de estados
NOP	No operación	· · X · X · · ·	00 000 000 00	1	1	4
HALT	Detiene el Z80	· · X · X · · ·	01 110 110 76	1	1	4
DI*	IFF←0	· · X · X · · ·	11 110 011 F3	1	1	4
EI*	IFF←1	· · X · X · · ·	11 111 011 FB	1	1	4
IM 0	Selecciona modo 0 para las interrupciones	· · X · X · · ·	11 101 101 ED 01 000 110 46	2	2	8
IM 1	Selecciona modo 1 para las interrupciones	· · X · X · · ·	11 101 101 ED 01 010 110 56	2	2	8
IM 2	Selecciona modo 2 para las interrupciones	· · X · X · · ·	11 101 101 ED 01 011 110 5E	2	2	8

NOTAS: (1) BCD: decimal codificado en binario.
IFF representa la báscula de habilitación de interrupciones.
CY representa el indicador de arrastre.
* indica que las interrupciones no son examinadas al final de DI o EL.



— La instrucción NOP es una instrucción que no realiza ninguna operación. Puede parecer a simple vista inútil, pero ocupa 1 byte y pierde tiempo al ejecutarse, lo cual puede ser útil en actividades de temporización, etc.

— La instrucción HALT detiene al Z-80. Es una instrucción de interés para el hardware, ya que detiene físicamente al microprocesador, el cual se pone en alta impedancia dejando los buses libres para que los utilicen otros dispositivos. Sólo se puede abandonar este estado de «hibernación» con una interrupción.

— La instrucción DI desactiva, es decir, inhibe las interrupciones mascarables. La instrucción EI realiza la operación opuesta.

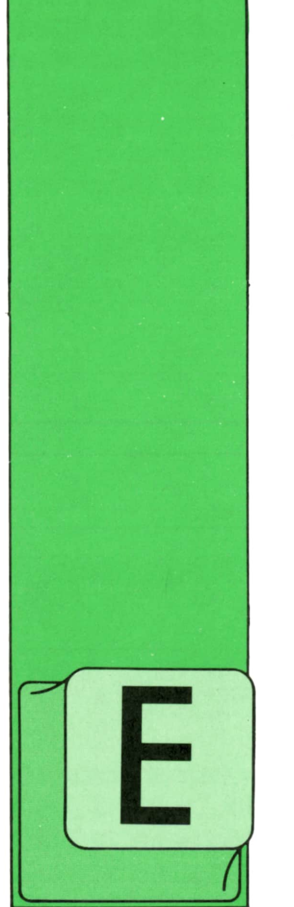
— Las instrucciones IM0, IM1 e IM2 seleccionan los tipos de interrupción mascarables 0, 1 y 2, respectivamente.

Estos tipos de interrupción y los demás serán tratados con mayor detalle en un apartado posterior.

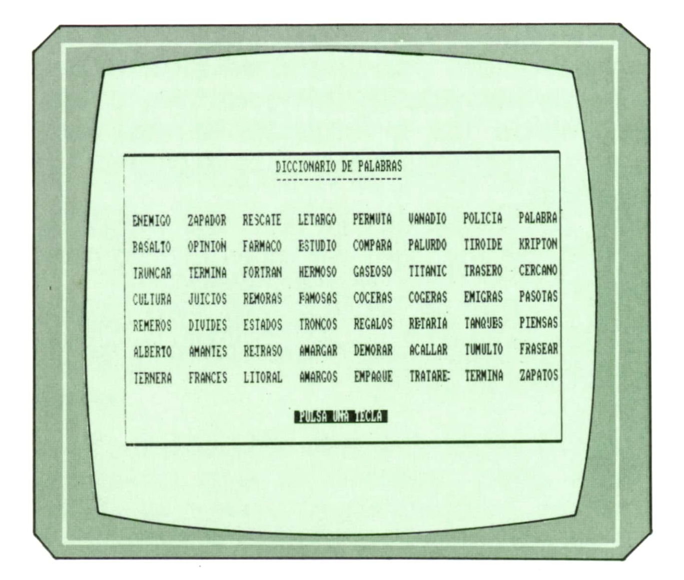
PROGRAMAS

EDUCATIVOS • DE UTILIDAD • DE GESTION • DE JUEGOS

Programa: Masterword para AMSTRAD



L programa que vamos a ver a continuación es una variación del famoso MASTER MIND numérico, pero con palabras. Este programa tiene su propio diccionario de palabras de siete letras y con él será con el que juguemos.

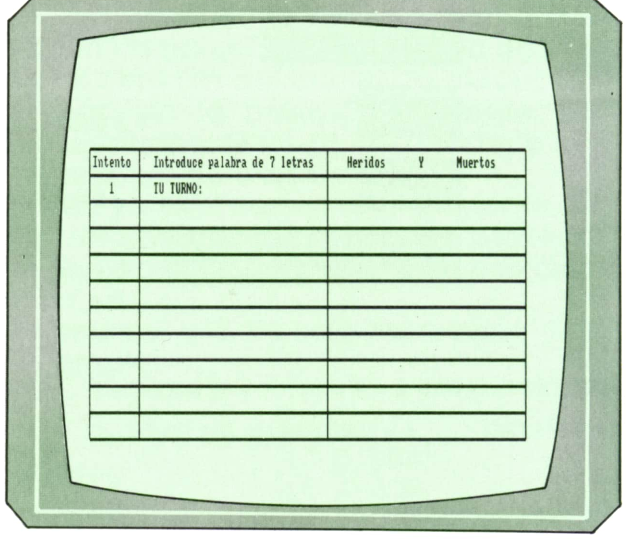




Estas son las palabras que podemos usar en el juego.

El juego consiste en adivinar la palabra que ha pensado el ordenador. Por supuesto, dicha palabra será una de las del diccionario.

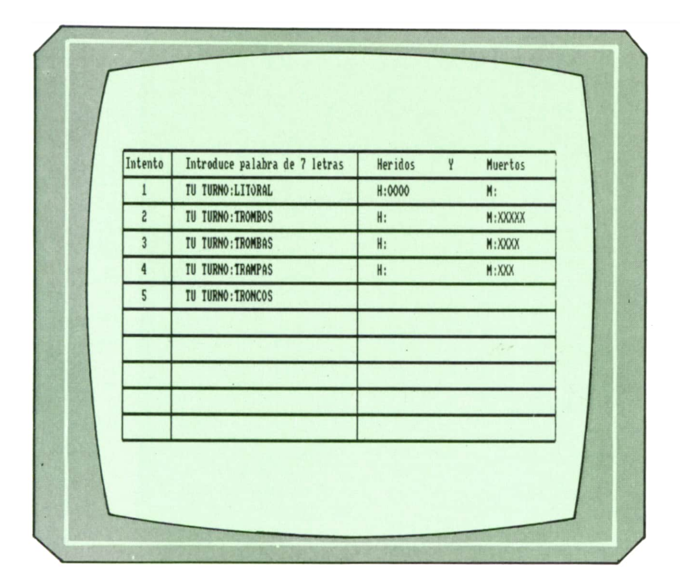
El jugador tiene que adivinar dicha palabra en un máximo de diez intentos. En caso de que no lo haga, el ordenador le dirá cuál era la palabra que había pensado.





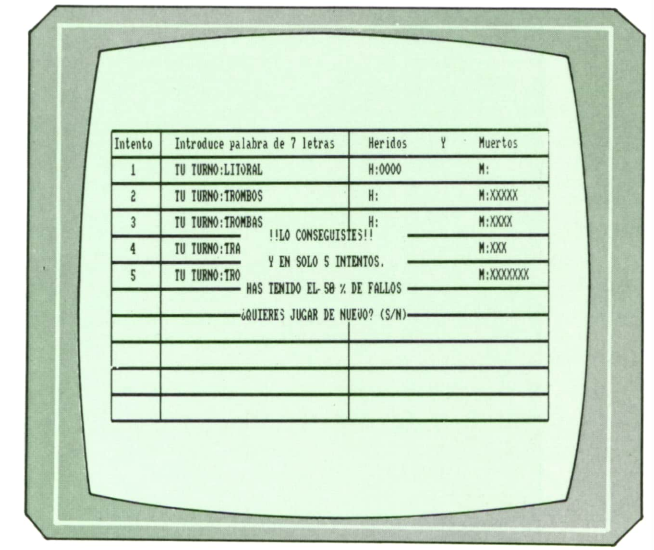
El tablero de juego.

Cada vez que el jugador realice una jugada, el ordenador le dirá cuántas letras de la palabra que ha introducido están bien colocadas y cuántas letras, aun no estando bien colocadas existen en la palabra. A las primeras letras, las que están colocadas en el mismo lugar que en la palabra que ha pensado el ordenador, las llamaremos MUERTOS. Las letras que, existiendo en la palabra a adivinar, el jugador las ha colocado en posiciones distintas, las llamaremos HERIDOS.



 *En medio de una partida.*

Cuando termina el juego, ganes o pierdas, el ordenador te dirá qué tanto por ciento de aciertos has tenido y te preguntará si quieres volver a jugar.

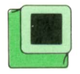



```
10 REM ****************************
20 REM ***       MASTERMVORD      ***
30 REM *** Un programa realizado ***
40 REM ***          Por           ***
50 REM *** Carlos A. Maria Morin ***
60 REM ***                        ***
70 REM ***      (C) Ediciones     ***
80 REM ***  Siglo Cultural 1987   ***
90 REM ****************************
100 REM
110 MODE 0
120 LOCATE 1,13
130 PRINT"M A S T E R W O R D"
140 FOR re=1 TO 3000
150 NEXT
160 MODE 2
170 CLEAR:CLS
180 REM
190 REM ************************************
200 REM ** PREPARA DICCIONARIO DE PALABRAS **
210 REM ************************************
220 REM
230 GOSUB 1600
240 LOCATE 29,3
250 PRINT"DICCIONARIO DE PALABRAS"
260 LOCATE 29,4
270 PRINT"----------------------"
280 RESTORE 2070
290 FOR y=7 TO 20 STEP 2
300 t-3
310 FOR x=1 TO 8
320 READ h$
330 LOCATE f,y
340 PRINT h$;
350 f=f+10
360 NEXT
370 NEXT
```

```
380 LOCATE 32,22
390 PRINT CHR$(24)" PULSA UNA TECLA "CHR$(24)
400 CALL &BB06:REM ESPERA QUE SE PULSE UNA TECLA
410 CLS
420 GOSUB 1600:' PREPARA VENTANA PRINCIPAL
430 GOSUB 1330:' PREPARA VENTANA PRINCIPAL
440 REM
450 REM ************************************
460 REM ** ELIGE PALABRA EN MODO ALEATORIO **
470 REM ************************************
480 REM
490 RANDOMIZE TIME
500 RESTORE 2070
510 FOR bu=1 TO INT(RND*57)
520 READ h$
530 NEXT
540 REM
550 REM ************************************
560 REM ******** PROGRAMA PRINCIPAL *********
570 REM ************************************
580 REM
590 LOCATE 2,3
600 PRINT"Intento"
610 LOCATE 13,3
620 PRINT"Introduce palabra de 7 letras"
630 LOCATE 48,3
640 PRINT"Heridos       Y       Muertos"
650 au=3
660 FOR i=1 TO 10
670 l=0:g=0
680 au=au+2
690 LOCATE 4,au
700 PRINT i
710 LOCATE 13,au
720 INPUT "TU TURNO:",a$
730 a$=UPPER$(a$)
740 IF LEN(a$)<7 OR LEN(a$)>7 THEN SOUND 1,100,10,15:GOTO 690
750 FOR bu1=1 TO 7
760 g$=MID$(a$,bu1,1)
770 d$=MID$(h$,bu1,1)
780 in(bu1)=ASC(g$)
790 re(bu1)=ASC(d$)
800 NEXT
810 FOR bu1=1 TO 7
820 IF in(bu1)=re(bu1) THEN l=l+1:GOSUB 1860
830 NEXT
840 FOR bu1=1 TO 7
850 in0=in(bu1)
860 GOSUB 1970
870 NEXT
880 g$=""
890 FOR ga=1 TO g
900 g$=g$+"O"
910 NEXT
920 l$=""
930 FOR ga=1 TO l
940 l$=l$+"X"
950 NEXT
960 LOCATE 48,au
970 PRINT "H:"g$
980 LOCATE 68,au
990 PRINT "M:"l$
1000 IF l=7 THEN GOTO 1160
1010 NEXT
1020 REM
1030 REM ************************************
1040 REM *** MENSAJES DE GANADOR/PERDEDOR ****
1050 REM ************************************
```

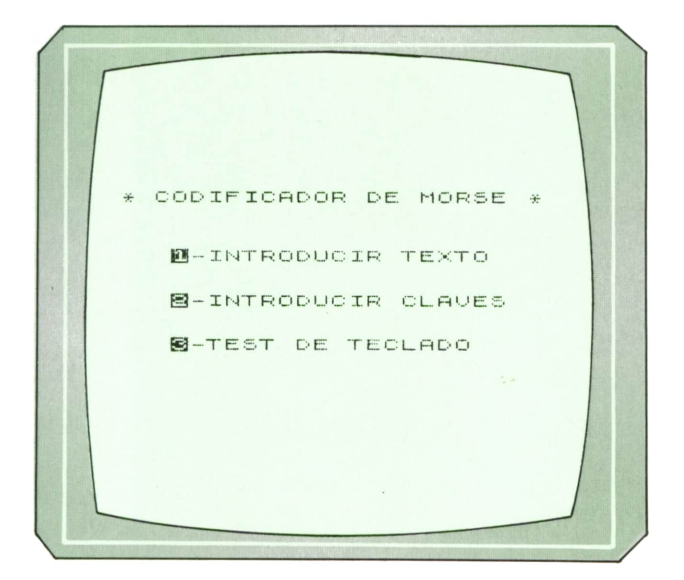
```
1060 REM
1070 FOR y=10 TO 16
1080 LOCATE 25,y
1090 PRINT"                              "
1100 NEXT
1110 LOCATE 30,10
1120 PRINT" NO LO CONSEGUISTES"
1130 LOCATE 26,13
1140 PRINT" LA PALABRA ES:    ";h$
1150 GOTO 1260
1160 FOR y=10 TO 16
1170 LOCATE 25,y
1180 PRINT"                              "
1190 NEXT
1200 LOCATE 30,10
1210 PRINT"!!LO CONSEGUISTES!!"
1220 LOCATE 30,12
1230 PRINT"Y EN SOLO";i;"INTENTOS."
1240 LOCATE 26,14
1250 PRINT"HAS TENIDO EL";i*10;"% DE FALLOS"
1260 LOCATE 25,16
1270 PRINT"]QUIERES JUGAR DE NUEVO? (S/N)"
1280 s$=UPPER$(INKEY$)
1290 IF s$<>"S" AND s$<>"N" THEN GOTO 1280
1300 IF s$="S" THEN GOTO 170
1310 CLS
1320 END
1330 REM
1340 REM ************************************
1350 REM ** SUBRUTINA PARA DIBUJAR VENTANA **
1360 REM ************************************
1370 REM
1380 FOR y=4 TO 23 STEP 2
1390 LOCATE 1,y
1400 PRINT CHR$(151);
1410 FOR x=1 TO 78
1420 PRINT CHR$(154);
1430 NEXT
1440 PRINT CHR$(157)
1450 NEXT
1460 va=34
1470 FOR x=10 TO 74 STEP va
1480 LOCATE x,2
1490 PRINT CHR$(158)
1500 FOR y=3 TO 23
1510 LOCATE x,y
1520 IF (y MOD 2)<>0 THEN PRINT CHR$(149) ELSE PRINT CHR$(159)
1530 va=27
1540 NEXT
1550 LOCATE x,y
1560 PRINT CHR$(155)
1570 NEXT
1580 RETURN
1590 REM
1600 REM ****** SEGUNDA SUBRUTINA ******
1610 REM
1620 LOCATE 1,2
1630 PRINT CHR$(150);
1640 FOR x=2 TO 79
1650 PRINT CHR$(154);
1660 NEXT
1670 PRINT CHR$(156)
1680 FOR y=3 TO 24
1690 LOCATE 1,y
1700 PRINT CHR$(149);
1710 LOCATE 80,y
1720 PRINT CHR$(149)
1730 NEXT
```

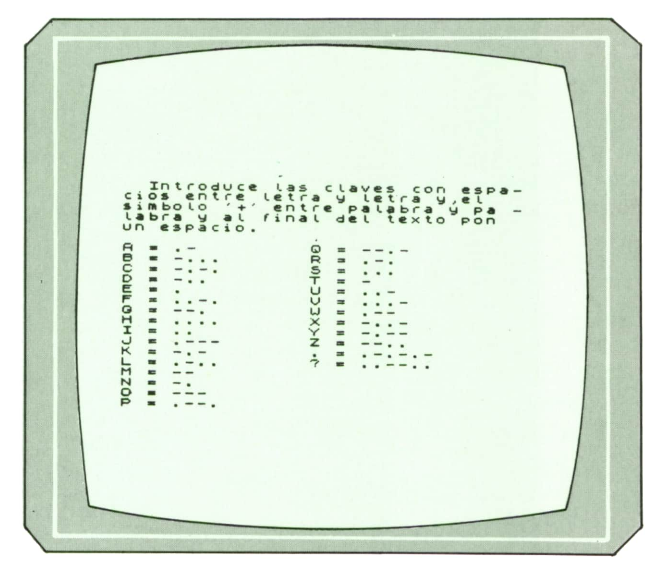
```
1740 LOCATE 1,24
1750 PRINT CHR$(147);
1760 FOR x=2 TO 79
1770 PRINT CHR$(154);
1780 NEXT
1790 PRINT CHR$(153)
1800 RETURN
1810 REM
1820 REM *************************************
1830 REM * SUBRUTINAS DEL PROGRAMA PRINCIPAL *
1840 REM *************************************
1850 REM
1860 in0=in(bu1)
1870 re0=re(bu1)
1880 GOSUB 1920
1890 in(bu1)=in0
1900 re(bu1)=re0
1910 RETURN
1920 tr=tr+1
1930 in0=400+tr
1940 tr=tr+1
1950 re0=400+tr
1960 RETURN
1970 FOR bu2=1 TO 7
1980 IF in0<>re(bu2) THEN GOTO 2020
1990 tr=tr+1
2000 g=g+1
2010 re(bu2)=400+tr
2020 NEXT
2030 RETURN
2040 REM
2050 REM ****** DATAS DE PALABRAS ******
2060 REM
2070 DATA ENEMIGO,ZAPADOR,RESCATE,LETARGO,PERMUTA,VANADIO,POLICIA,PALABRA
2080 DATA BASALTO,OPINION,FARMACO,ESTUDIO,COMPARA,PALURDO,TIROIDE,KRIPTON
2090 DATA TRUNCAR,TERMINA,FORTRAN,HERMOSO,GASEOSO,TITANIC,TRASERO,CERCANO
2100 DATA CULTURA,JUICIOS,REMORAS,FAMOSAS,COCERAS,COGERAS,EMIGRAS,PASOTAS
2110 DATA REMEROS,DIVIDES,ESTADOS,TRONCOS,REGALOS,RETARIA,TANQUES,PIENSAS
2120 DATA ALBERTO,AMANTES,RETRASO,AMARGAR,DEMORAR,ACALLAR,TUMULTO,FRASEAR
2130 DATA TERNERA,FRANCES,LITORAL,AMARGOS,EMPAQUE,TRATARE,TERMINA,ZAPATOS
```



Programa: Lector de Morse para Spectrum

Con este pequeño programa podremos aprender *Morse* de una forma rápida y muy sencilla. El programa nos permite meter texto con letras y nos lo pasa a líneas de puntos. También nos deja entrar directamente código *Morse* y nos lo traduce para que veamos si lo hemos escrito bien. Como es lógico, y para que vayamos cogiendo la práctica necesaria, tenemos una opción del menú general para practicar con el teclado e ir ganando velocidad.






Menú general del programa.

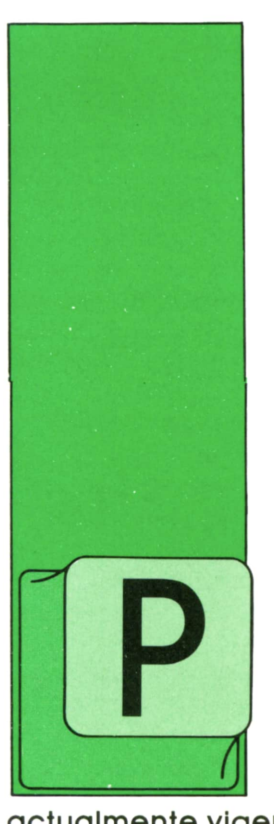

Pantalla que aparece cuando queremos introducir código Morse.

```
 10 REM ***********************
 20 REM *** LECTOR DE MORSE ***
 30 REM ***********************
 40 REM ** POR CARLOS DORAL ***
 50 REM ***********************
 55 REM
 60 DIM c$(28,6)
 70 FOR f=1 TO 26
 80    READ c$(f)
 90 NEXT f
100 REM  CLAVES DE LAS LETRAS
110 DATA "130000"
120 DATA "311100"
130 DATA "313100"
140 DATA "311000"
150 DATA "100000"
160 DATA "113100"
170 DATA "331000"
180 DATA "111100"
190 DATA "110000"
200 DATA "133300"
210 DATA "313000"
220 DATA "131100"
230 DATA "330000"
240 DATA "310000"
250 DATA "333000"
260 DATA "133100"
270 DATA "331300"
280 DATA "131000"
290 DATA "111000"
300 DATA "300000"
310 DATA "113000"
320 DATA "111300"
330 DATA "133000"
340 DATA "311300"
350 DATA "313300"
360 DATA "331100"
370 DATA "131313"
380 DATA "113311"
```

```
390 LET co=1
400 CLS
410 PRINT AT 0,3; FLASH 1;" * CODIFICADOR DE MORSE * "
420 PRINT AT 4,7;"1-INTRODUCIR TEXTO"
430 PRINT AT 7,7;"2-INTRODUCIR CLAVES"
440 PRINT AT 10,7;"3-TEST DE TECLADO"
450 LET we=100+(co*10)
455 RESTORE we
460 LET a$=c$(co)
470 LET rn=INT (RND*5)+1
480 LET a=VAL a$(rn TO rn)
490 BEEP a*.10,10
500 LET co=co+1: IF co=26 THEN  LET co=1
510 LET k$=INKEY$
520    IF k$="1" THEN  GO SUB 560: GO TO 390
530    IF k$="2" THEN  GO SUB 780: GO TO 390
540    IF k$="3" THEN  GO SUB 1100: GO TO 390
550 GO TO 460
555 REM
560 REM ***********************
570 REM ** INTRODUCIR TEXTO ***
580 REM ***********************
585 REM
590 CLS
600 POKE 23658,8
610 INPUT "Introduce texto= ";a$
620 FOR f=1 TO LEN a$
630    LET b$=a$(f TO f)
640    LET n=CODE b$-64
650    PRINT b$;
660    IF b$=" " THEN  FOR p=1 TO 15: NEXT p: NEXT f
670    LET b$=c$(n)
680    FOR c=1 TO 6
690      LET b=VAL b$(c TO c)
700      BEEP b*.10,10
710    NEXT c
720 NEXT f
730 PRINT #0;"YA ESTA,QUIERES OTRO TEXTO(S/N)?"
740 LET k$=INKEY$
750    IF k$="S" THEN  GO TO 560
760    IF k$<>"S" AND k$<>"" THEN  RETURN
770 GO TO 740
775 REM
780 REM ***********************
790 REM ** INTRODUCIR CLAVES **
800 REM ***********************
805 REM
810 CLS
820 PRINT "  Introduce las claves con espa-cios entre letra y letra y,el   simb
olo '+' entre palabra y pa -labra y al final del texto pon  un espacio."
830 PRINT
840 LET y=6
850 LET x=0
860 LET c=65
870 FOR f=1 TO 26
880    PRINT AT y,x;CHR$ c;" = ";
890    LET b$=c$(f)
900    FOR i=1 TO 6
910      IF b$(i TO i)="1" THEN  PRINT ".";
920      IF b$(i TO i)="3" THEN  PRINT "-";
930    NEXT i
940    LET c=c+1
950    IF f=16 THEN  LET y=5: LET x=x+15: PRINT AT y,x
960    LET y=y+1
970 NEXT f
980 PRINT AT y,x;". = .-.-.-"
990 PRINT AT y+1,x;"? = ..--.."
```

```
1000 INPUT "Introduce las claves ";a$
1010 CLS
1020 LET s$=""
1030 FOR f=1 TO LEN a$
1040    LET s$=s$+a$(f TO f)
1050    IF a$(f TO f)=" " OR a$(f TO f)="+" THEN  GO SUB 1260: LET s$=""
1060 NEXT f
1070 PRINT #0;" Pulsa una tecla para el menu."
1080 IF INKEY$="" THEN  GO TO 1080
1090 RETURN
1095 REM
1100 REM ***********************
1110 REM *** TEST DE TECLADO ***
1120 REM ***********************
1125 REM
1130 CLS
1140 PRINT "   Pulsa la tecla 'g' para pro-  ducir los sonidos"
1150 FOR f=1 TO 100
1160 NEXT f
1170 PRINT ' FLASH 1;" PULSA UNA TECLA PARA COMENZAR "
1180 IF INKEY$="" THEN  GO TO 1180
1190 CLS
1200 LET co=0
1210    IF INKEY$="g" OR INKEY$="G" THEN  PRINT "-";: BEEP .30,10
1220    IF INKEY$="h" OR INKEY$="H" THEN  PRINT ".";: BEEP .10,10
1230    IF INKEY$=" " THEN  PRINT " ";: FOR f=1 TO 20: NEXT f
1240    IF INKEY$=CHR$ 13 THEN  RETURN
1250 GO TO 1210
1255 REM
1260 REM ***********************
1270 REM **** busca claves *****
1280 REM ***********************
1285 REM
1290 LET w$=""
1300 FOR c=1 TO LEN s$-1
1310    IF s$(c TO c)="-" THEN  LET w$=w$+"3"
1320    IF s$(c TO c)="." THEN  LET w$=w$+"1"
1330    IF s$(c TO c)="+" THEN  PRINT " ";: RETURN
1340 NEXT c
1350 LET q$="000000"
1360 LET w$=w$+q$( TO 7-LEN s$)
1370 FOR c=1 TO 28
1380    IF w$=c$(c) THEN  PRINT CHR$ (64+c);: IF a$(f TO f)="+" THEN  PRINT " '
: RETURN
1390 NEXT c
1400 RETURN
```

TECNICAS DE ANALISIS



Métodos predefinidos de desarrollo

PARECE útil considerar a continuación algún método general predefinido de análisis, desarrollo e implementación de aplicaciones informáticas.

Entre los métodos actualmente vigentes de conducción de proyectos informáticos, uno de los más utilizados es el MERISE, concebido por un grupo de empresas consultoras y de desarrollo de software francesas bajo los auspicios del Ministerio de Industria y de la «Misión para la Informática» de aquel país.

Por supuesto, la mayoría de las grandes compañías fabricantes de ordenadores utilizan habitualmente sus propios sistemas de desarrollo o metodologías externas adoptadas como estándar, pero entendemos que entre las metodologías desarrolladas por grupos independientes con finalidad de servir de estándar, una de las más generales y de las más utilizadas hoy en día (y cada día más, según parece) es MERISE.

Antes de entrar a definir el método en sus detalles parece útil hacer algunas reflexiones sobre la importancia de utilizar algún método predefinido en el desarrollo de las aplicaciones informáticas, en una organización cualquiera.

En efecto, hoy en día se estudian cada vez con más detalle las inversiones realizadas en el área de la informática. En la actualidad pocos empresarios hay que sigan considerando que con la mera compra de un ordenador van a resolver ningún problema en la empresa y que, por tanto, dicha compra es una inversión grande, pero útil. Por el contrario, se considera que la implantación de un sistema informático (hardware, más software) puede suponer cambios de modo de trabajo, modificación de rutinas (y de mentalidades en ocasiones) y puede tener un enorme efecto (para bien o para mal) en su organización: cada día más se considera que un sistema informático más que un programa concreto de ordenador es todo un modo de razonamiento, una cierta «cultura», un saber-hacer de la empresa e, incluso a veces, refleja un estilo gerencial y un estilo de organización. Lo que nadie duda es que más que problemas técnicos, normalmente, la implantación de un sistema informático suele comportar problemas humanos y de organización que deben ser abordados de un modo global.

Por otro lado, la importancia en pesetas de cualquier decisión tomada en este área suele ser grande por lo que ha de ser meditada y cimentada en estudios consistentes y completos. En efecto, se puede calcular, por ejemplo, que la aplicación de facturación de una gran empresa comercial que tenga una cifra de negocios de algunos miles de millones de pesetas o el control de producción de una factoría de 3.000 personas puede suponer una inversión de unos 60 millones de pesetas (estimando unas 80.000 líneas de programación y un coste de unas 750 pesetas/línea).

Unos costes de esta envergadura (se ha calculado que, en una empresa bien dirigida, con una distribución de costes normal, el presupuesto informático supone un 2 por 100 del volumen de negocio total) exige, desde luego, una estructura de control y una planificación adecuadas.

Existen otras razones que avalan la adopción de una tecnología concreta

(un método) para el desarrollo de las aplicaciones informáticas. Por un lado, hay que tener en cuenta la movilidad de las necesidades informáticas de una organización: es muy común que antes de que un sistema de proceso de personal, de gestión de la producción, etc., haya sido puesto a punto, las normas bajo las que se trabajaba y con las que se ha constituido el sistema, hayan cambiado. Normalmente, también, los diferentes procesos han sido desarrollados e instalados por grupos de profesionales distintos y sólo después de que hayan sido probados se decide su integración o su interconexión. Incluso, sin que los procesos o sistemas se comuniquen, hay movilidad de las personas (que cambian de grupo de trabajo o abandonan la organización y deben ser sustituidas). De hecho, en cualquier Servicio Informático con algunos años dentro de cualquier organización, se dedica hasta un 40 por 100 (y más en ocasiones) de la capacidad total de trabajo al mantenimiento de aplicaciones y sistemas ya en funcionamiento. Es decir, con un buen método de concepción y gestión de proyectos informáticos hay que conseguir que los expertos dediquen su tiempo a cómo resolver los problemas presentados y no cómo estudiarlos o cómo documentar las soluciones diseñadas. Además, hay que ofrecer a toda la organización un lenguaje común válido para diferentes grupos informáticos (empeñados en tareas diversas dentro de la organización), para los grupos de organización y hasta para los usuarios interesados.

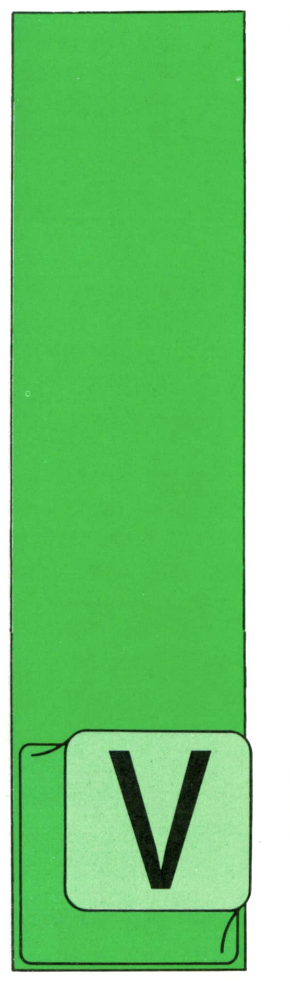
Es importante distinguir también el método de las «herramientas» utilizadas en su implementación. En efecto, últimamente se han desarrollado enormemente diversas «herramientas informáticas» que ayudan enormemente en el desarrollo de procesos: existen lenguajes evolucionados muy cómodos de ser usados (verdaderos metalenguajes) y lenguajes de «generación de aplicaciones» (normalmente se alude a este tipo de lenguajes como de cuarta generación); otros lenguajes ofrecen un diccionario de datos, con lo que los sistemas se construyen sobre este diccionario y se consigue gran coherencia en los datos y eliminación de las redundancias; se comercializan también «cajas de herramientas» para la ayuda a la concepción de aplicaciones, etcétera.

Con el auxilio de estas «herramientas» se ha constatado que se puede aumentar de tres a cinco veces la producción dada de un grupo (medida en volumen de programas o líneas de programación/año escritas).

Sin embargo, el concepto de «método de desarrollo» es más general y se refiere a la concepción y diseño del sistema y a las técnicas de control de su implementación, independientemente de que luego en la producción propiamente dicha se utilicen unas u otras «herramientas».

Se han desarrollado numerosos métodos de ayuda al diseño, pero la mayoría se refieren a la formalización de aspectos concretos o a la resolución de problemas particulares: métodos de concepción y diseño de bases de datos, métodos de gestión del proyecto informático, métodos de programación estructurada, métodos de análisis modular de sistemas: el método MERISE, por el contrario, pretende ser un método general que sirva de referencia a todos los elementos involucrados en la tarea del desarrollo del sistema informático y como «cuerpo metodológico» completo y coherente para la armonización de todas las actividades a desarrollar.

TECNICAS DE PROGRAMACION



Instrucciones en bucle (continuación)

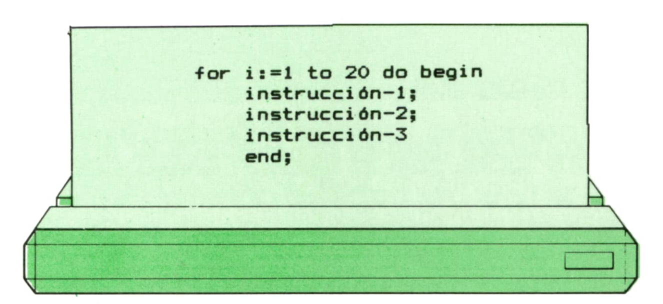
VEAMOS cómo se construye en PASCAL la instrucción de bucle que permite ejecutar una instrucción cierto número de veces (por ejemplo, 20):

```
for i:=1 to 20 do instrucción;
```

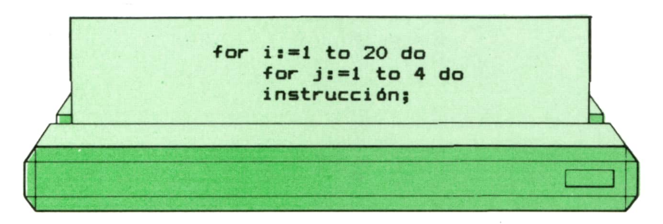
donde la variable de control del bucle (i en nuestro ejemplo) tiene que tener tipo entero.

Al revés que en BASIC, donde se precisan tres instrucciones, en PASCAL basta con una, cuya traducción literal es la misma: «Para i = 1 hasta 20, ejecutar la instrucción y pasar al siguiente valor de i». Su organigrama es, naturalmente, el mismo que el de la estructura BASIC correspondiente.

En BASIC, el hecho de que esta forma de instrucción de bucle se divida en tres partes (inicio del bucle, instrucción a ejecutar y fin del bucle) permite ampliarla con facilidad al caso en que deseamos ejecutar varias instrucciones dentro del mismo bucle. En PASCAL esto no es posible, pues no existe marca de fin de bucle. Sin embargo, disponemos del bloque secuencial de instrucciones, que podemos utilizar como instrucción de bucle sin dificultades:



Tampoco tendremos problema para incluir dentro de un bucle asignaciones de valor, instrucciones condicionales u otros bucles, incluso de este mismo tipo. Sin embargo, al igual que en BASIC, si se incluye un bucle FOR dentro de otro bucle FOR, es necesario utilizar una variable diferente para cada uno.



Veamos ahora cómo puede utilizarse la instrucción FOR para modificar el programa PASCAL que ordena tres números en orden ascendente, de tal manera que pueda aplicarse a la ordenación de un número de valores diferente.

```
program ordenar;
var
   n,i:integer;
   x:boolean;
   y:real;
   a:array[1..100] of real;
begin
   write ('Número de datos a ordenar: ');
   readln (n);
   writeln ('Deme ',n,' datos');
   for i:=1 to n do
       readln (a[i]);
   x:=true;
   while x do begin
      x:=false;
      for i:=1 to n-1 do
         if a[i]>a[i+1] then begin
            y:=a[i+1];
            a[i+1]:=a[i];
            a[i]:=y;
            x:=true
         end
   end;
   for i:=1 to n do
       writeln (a[i])
end.
```

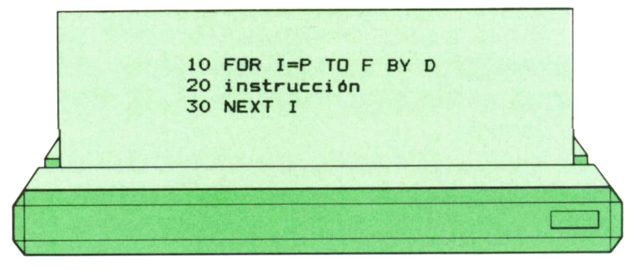
El programa anterior funciona exactamente igual que la versión BASIC que vimos al final del capítulo anterior. Tan sólo incluye una pequeña mejora: antes de pedir datos del terminal escribe un mensaje que indica qué es lo que espera el programa que le proporcionemos.

Veamos un ejemplo de su funcionamiento:

```
RUN
Número de datos a ordenar: 4
Deme 4 datos
2
30
-1
0
-1.00000000E+000
 0.00000000E+000
 2.00000000E+000
 3.00000000E+001
```

En APL tampoco existe esta forma de la instrucción de bucle. Ya vimos en el capítulo anterior que es posible construir nuestro programa de ordenación de datos sin necesidad de bucle alguno, por lo que este tipo de instrucciones no es tan necesario en este lenguaje. Sin embargo, es posible simularlas por medio de la instrucción de transferencia condicional, como veremos más adelante.

La estructura de bucle que ejecuta cierto número de veces una instrucción o grupo de instrucciones determinado presenta ciertas variantes, tanto en BASIC como en PASCAL. En la forma que hemos visto hasta ahora, la variable de control de bucle adopta inicialmente el valor 1 y aumenta una unidad por cada vuelta del bucle. En el caso más general posible, en el primero de estos lenguajes es posible modificar el valor inicial y el incremento, haciéndolos tomar cualquier valor. Para ello se utiliza la siguiente forma de la instrucción FOR:

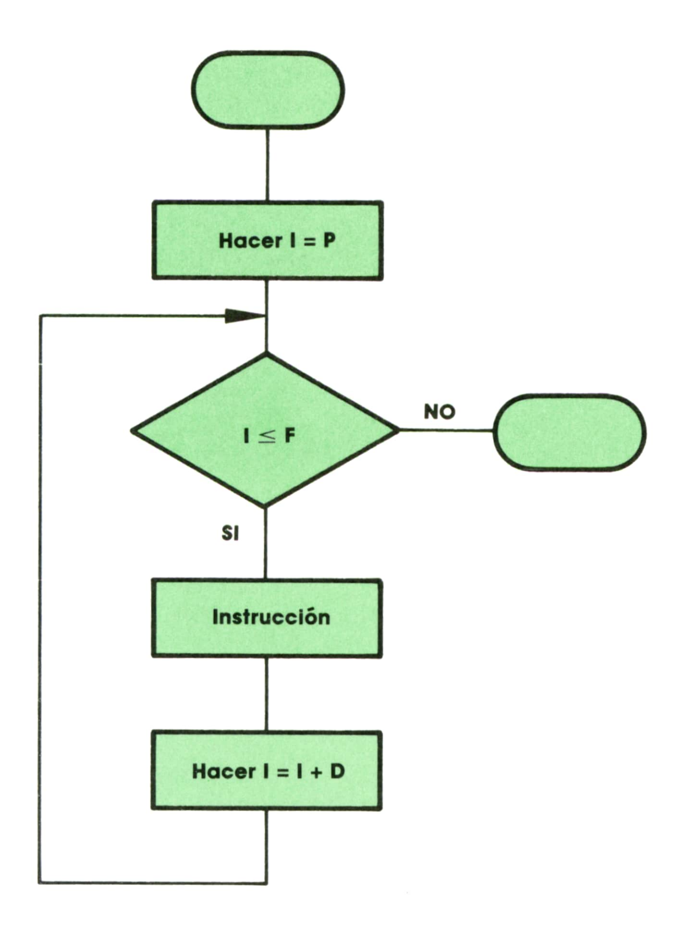


```
10 FOR I=P TO F BY D
20 instrucción
30 NEXT I
```

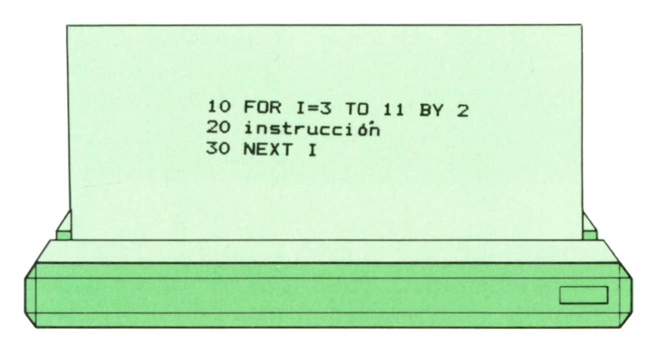
donde P es el valor de partida que debe asignarse a la variable de control del bu-

cle (I en este caso), F es el valor final que debe adoptar I y D es el incremento que tendrá que sumársele al valor de I al final de cada vuelta del bucle.

El organigrama de esta forma de la instrucción de bucle es el siguiente:



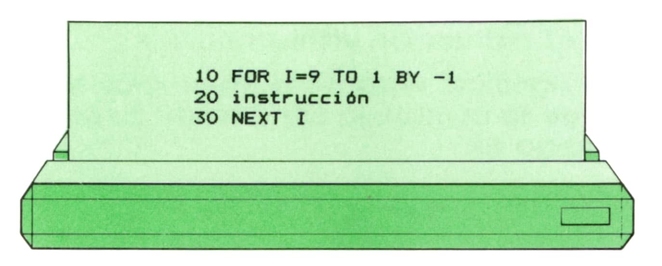
Veamos un ejemplo concreto:



```
10 FOR I=3 TO 11 BY 2
20 instrucción
30 NEXT I
```

donde I toma, sucesivamente, los valores 3, 5, 7, 9 y 11.

No hay nada que impida que el incremento sea negativo, lo que haría que los valores de la variable de control fueran decrecientes. Por ejemplo:

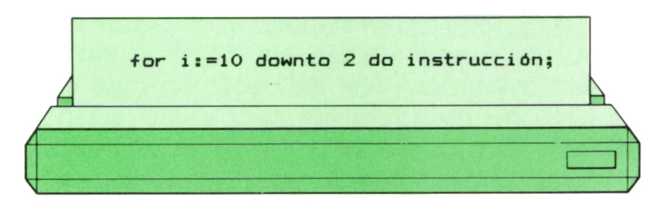


```
10 FOR I=9 TO 1 BY -1
20 instrucción
30 NEXT I
```

que ejecuta la instrucción nueve veces, para I = 9, 8, 7, 6, 5, 4, 3, 2, 1.

En PASCAL, también es posible cambiar el valor inicial del bucle, haciéndolo distinto de 1. Sin embargo, no se puede cambiar el valor del incremento. Lo que sí se puede es conseguir que los valores que toma sucesivamente la variable de control del bucle sean decrecientes, pero siempre han de ser consecutivos.

Veamos un ejemplo de un bucle PASCAL cuya variable toma valores decrecientes



```
for i:=10 downto 2 do instrucción;
```

donde la variable de control toma los valores i = 10, 9, 8, 7, 6, 5, 4, 3 y 2.

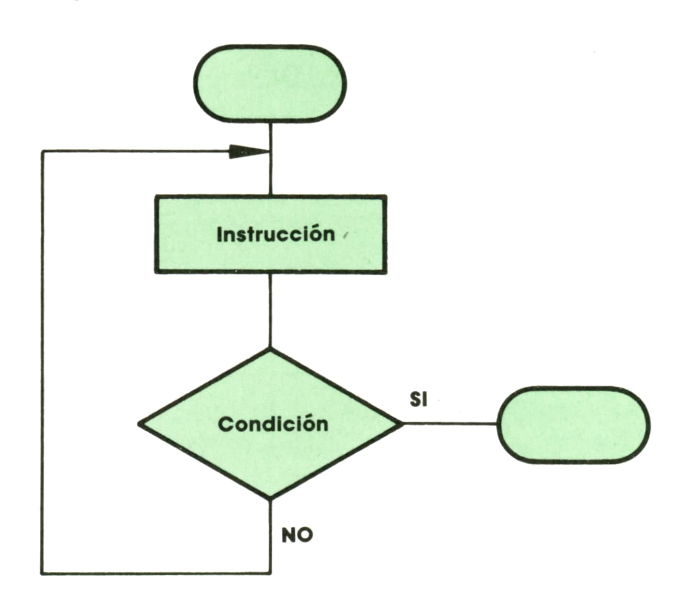
Tanto el punto de partida y el final del bucle (en BASIC y PASCAL) como el incremento (sólo en BASIC) pueden sustituirse por variables o por expresiones. Ya hemos visto un ejemplo de esto en el programa de ordenación incluido en este capítulo y el anterior.

En PASCAL existe aún una tercera forma de la instrucción de bucle, que normalmente no se presenta en BASIC y que se llama «estructura REPEAT». Es muy parecida a la estructura WHILE, pero se diferencia de ella en que la comprobación de la condición de salida del bucle se lleva a cabo después de que la instrucción contenida en el bucle ha sido ejecutada, y no antes. Esto significa que dicha instrucción será ejecutada al menos una vez, mientras que en el caso de la estructura WHILE es perfectamente posible que la instrucción contenida en el bucle no se ejecute nunca, si la condición de salida del bucle no se cumplía cuando el ordenador llegó a éste por primera vez.

Veamos cómo se programa la estructura REPEAT:

REPEAT instrucción UNTIL condición;

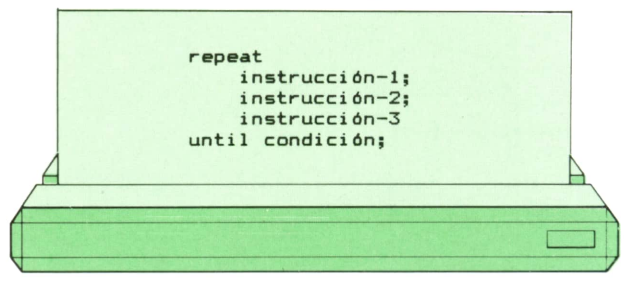
que significa: «Repetir la instrucción hasta que se cumpla la condición»: Su organigrama es:



En el organigrama se observa claramente que la instrucción interior del bucle se ejecuta al menos una vez.

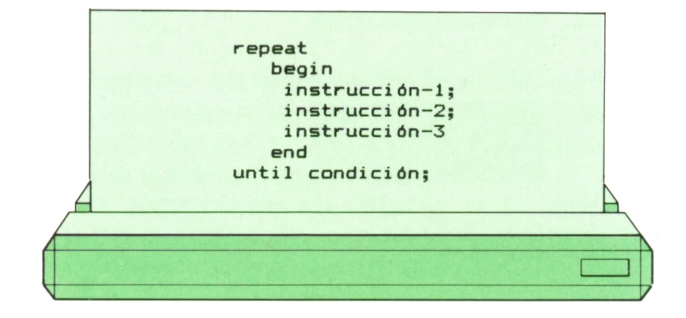
Al igual que en la estructura WHILE, es posible construir estructuras REPEAT que se conviertan en bucles permanentes, que no tengan salida, por lo que hay que tener un cuidado especial al programarlas.

En este caso, y al revés que en la estructura de WHILE de PASCAL, pero al igual que en la estructura WHILE de BASIC, se observará que la instrucción interior del bucle se encuentra emparedada entre dos palabras reservadas (REPEAT y UNTIL). Por tanto, si deseamos introducir dentro del bucle más de una instrucción, no es necesario utilizar un bloque secuencial delimitado por BEGIN y END, sino que basta escribirlas una a continuación de la otra, como en el siguiente ejemplo:



```
repeat
    instrucción-1;
    instrucción-2;
    instrucción-3
until condición;
```

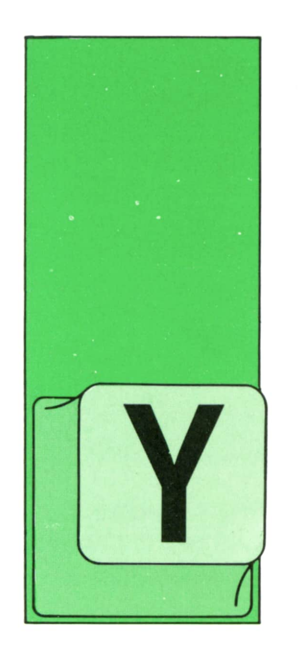
Naturalmente, si deseamos utilizar a pesar de todo la forma normal del bloque secuencial, no hay nada que nos lo impida:



```
repeat
   begin
    instrucción-1;
    instrucción-2;
    instrucción-3
   end
until condición;
```

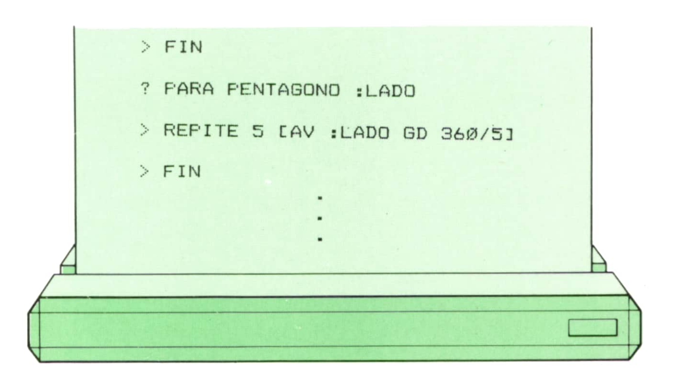
LOGO

Cómo dibujar cualquier polígono



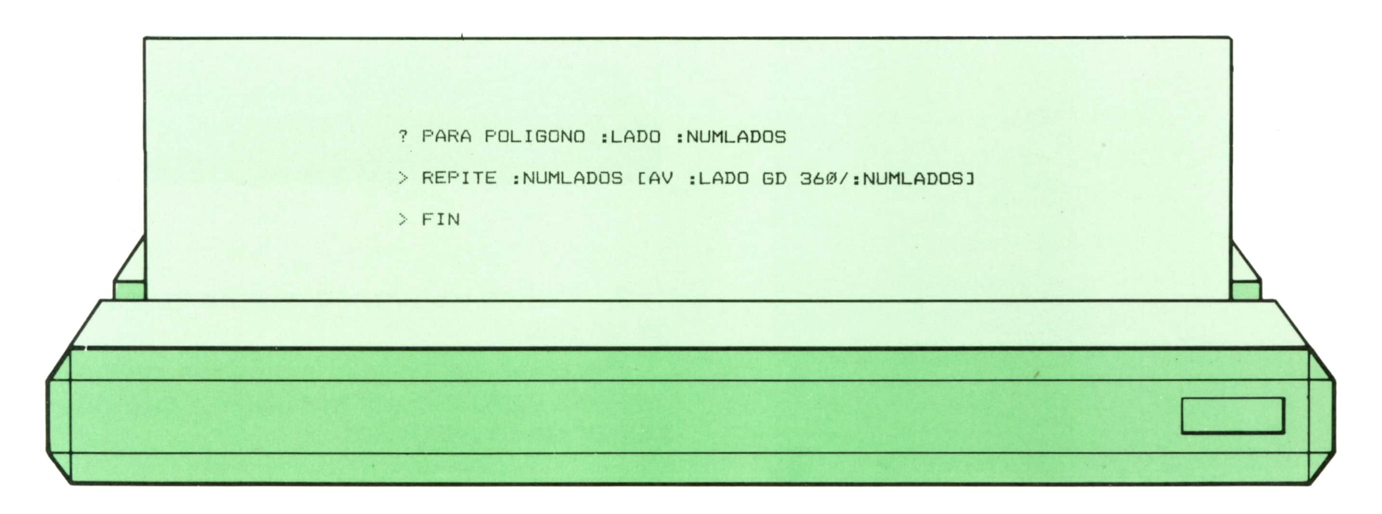
A vimos cómo podemos dibujar cualquier polígono regular. Para ello, teníamos que definirnos varios procedimientos:

```
? PARA TRIANGULO :LADO
> REPITE 3 [AV :LADO GD 360/3]
> FIN
? PARA CUADRADO :LADO
> REPITE 4 [AV :LADO GD 360/4]
```



Como se puede ver, hemos puesto una variable para que puedan ser de cualquier tamaño.

Si añadimos una variable más, tenemos la posibilidad de poder hacer todas estas figuras con un solo procedimiento. Con el valor que guardamos en esta variable le diremos a la tortuga el número de lados que va a tener el polígono que queremos dibujar. Para ello, pondremos:

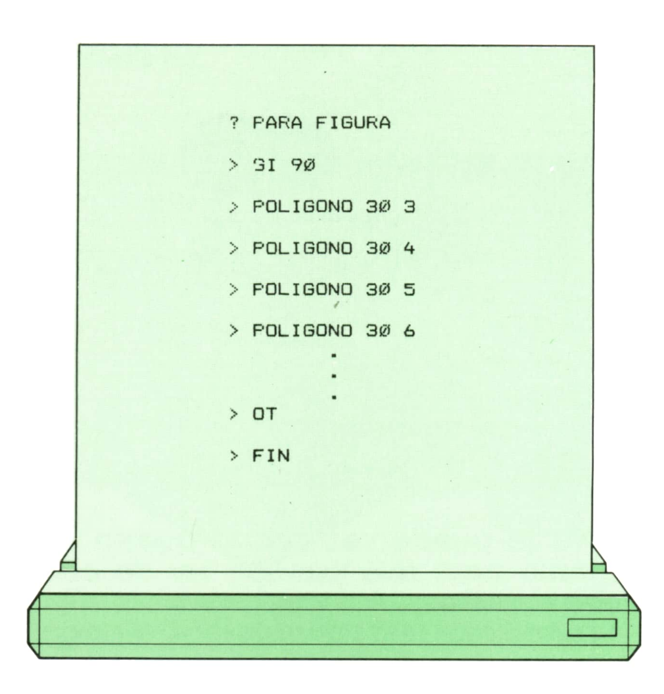


Ahora nos basta con ir dando diferentes valores a "NUMLADOS para obtener un triángulo (3), un cuadrado (4), un pentágono (5)...

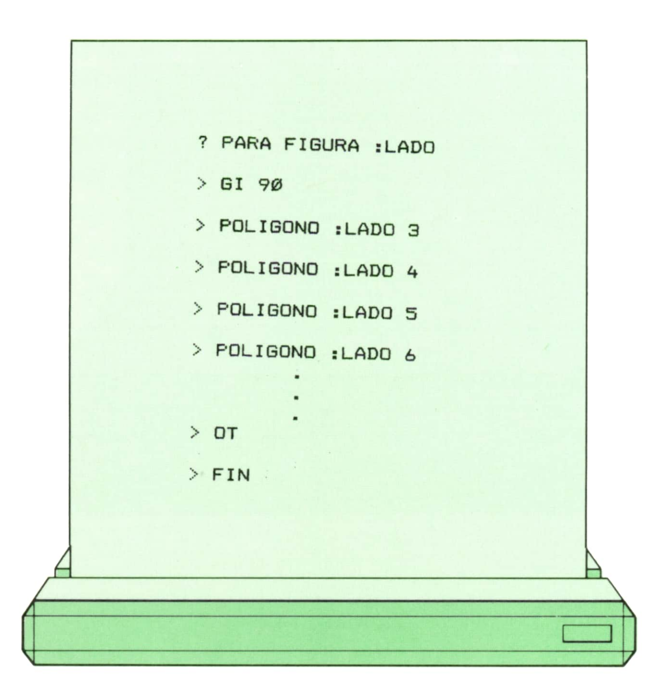
Podemos usar este procedimiento para hacer la siguiente figura:



escribiendo:

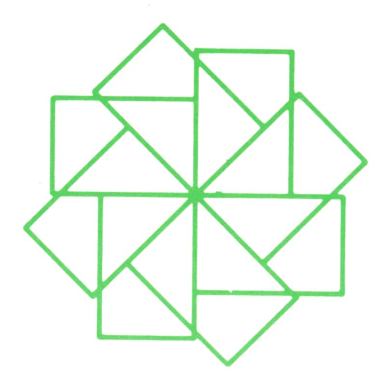


Si queremos que la longitud del lado de todos los polígonos no sea fija añadiríamos una variable a este procedimiento:

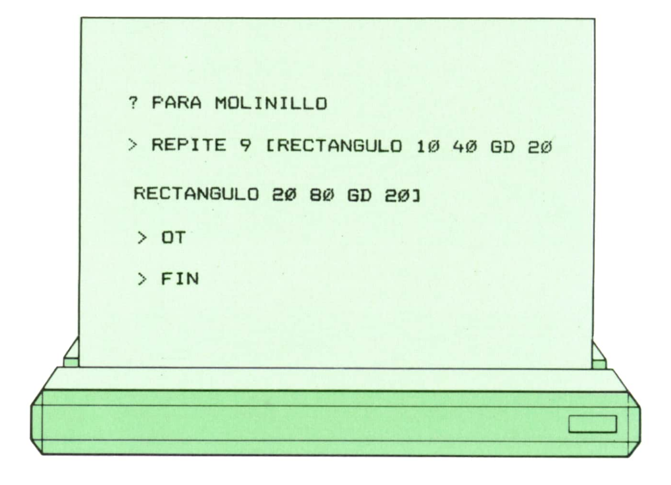


Os proponemos

1. Intenta dibujar el siguiente molinillo con la posibilidad de que pueda tener distintos tamaños:

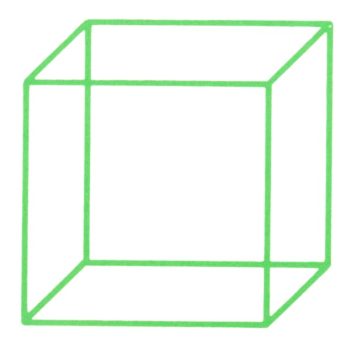


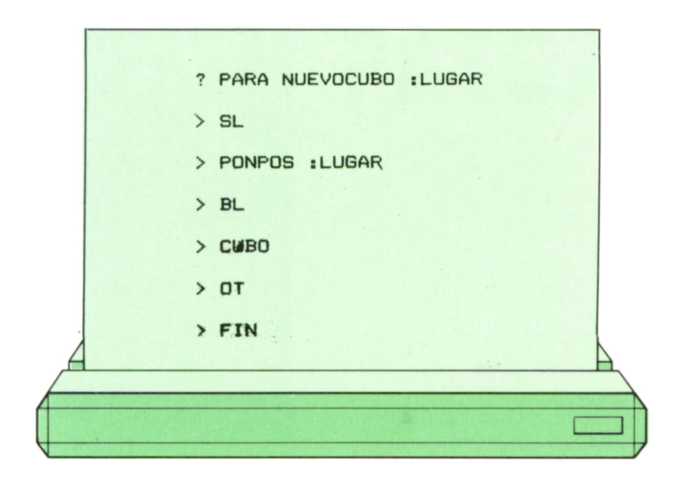
Cuando lo hayas hecho, puedes probar este otro procedimiento para dibujar otro molinillo en el que se mezclan rectángulos de dos tamaños.



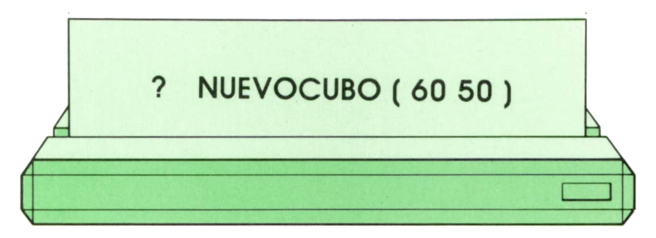
2. Define un procedimiento que dibuje un cubo:

Después de usar el siguiente procedimiento para dibujar el cubo en cualquier lugar de la pantalla:



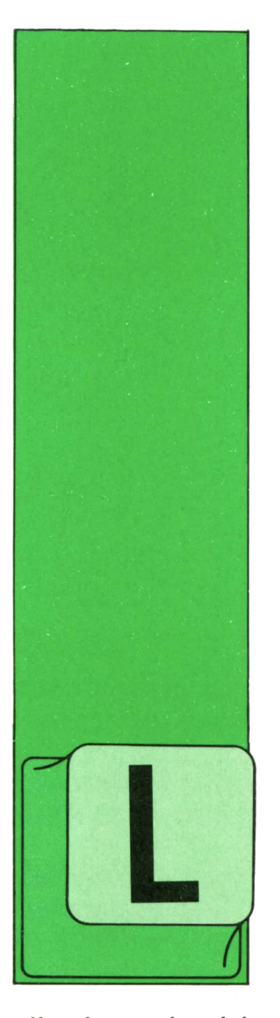


Para ejecutarlo, tendrás que dar a la variable un valor que sea una posición. Por ejemplo, con



la tortuga dibujará el cubo en la parte de arriba y a la derecha de la pantalla.

PASCAL





Registros

LOS registros o fichas (record en inglés) se utilizan en multitud de actividades de la vida diaria. En un banco puede que se utilicen para guardar el nombre de cada cliente y el saldo de sus diferentes cuentas; en una central lechera, para el número, fecha y destino de los diferentes lotes de leche; en una biblioteca, para el título, autor, editorial y estante de cada libro, etc.

Como se ve en estos ejemplos, podríamos describir una ficha como un conjunto de datos, no necesariamente del mismo tipo, que, al tener entre sí alguna clase de vínculo especial, se agrupan para formar una unidad.

Actualmente, cada vez se utilizan más los ordenadores para manejar este tipo de cosas. Sin ellos los bancos, por poner un ejemplo, serían incapaces de gestionar la enorme cantidad de cuentas existentes de una manera rápida y eficiente.

Del PASCAL, por ahora, sólo conocemos el tipo ARRAY para guardar diferentes cosas en una misma variable, pero todas han de ser del mismo tipo; sin embargo, las fichas pueden tener información muy variada: el nombre del titular de una cuenta sería un dato de tipo ARRAY OF CHAR, mientras que el saldo sería de tipo REAL. Como es de suponer, el PASCAL también permite manejar fichas.

Las diferentes partes de un registro se denominan CAMPOS; así, las fichas de la central lechera tendrían tres campos para los tres diferentes datos de que constan. En general, para definir el tipo de ficha necesario escribiríamos:

```
type
  Ficha_t = record
              Campo1: Tipo1;
              Campo2: Tipo2;
              ...
            end;
```

es decir, entre las palabras reservadas RECORD y END, y separados por punto y coma, se deben escribir los nombres de los diferentes campos seguidos de dos puntos y del tipo de dato correspondiente. Cuando dos campos o más son del mismo tipo, se puede ahorrar escritura poniendo sus nombres uno detrás de otro separados por comas, tras lo cual vendrían los dos puntos y el tipo. Como sucede en otros casos, es posible definir una variable de tipo ficha poniendo a su lado directamente la descripción, pero siempre es mejor definir antes el tipo.

Los campos pueden ser de CUALQUIER tipo o subrango de tipo que esté previamente definido.

Supongamos que queremos guardar fechas en variables de tipo registro. Estas tendrían tres campos: los de día y año, que serían números enteros, y el mes, que sería de tipo Mes_t:

```
type
  Mes_t = (Enero,Febrero,Marzo,Abril,Mayo,Junio,Julio,Agosto,
           Septiembre,Octubre,Noviembre,Diciembre);

  Fecha_t = record
              DiaMes,
              Anyo    : integer;
              Mes     : Mes_t
            end;
var
  FechaDeHoy,
  Nacimiento : Fecha_t;
```

Como sucedía en las variables de tipo ARRAY, la única operación posible con todos los elementos de un registro a la vez es la asignación:

Nacimiento := FechaDeHoy;

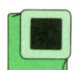
Esto copiaría los campos de FechaDeHoy, uno por uno, en los de Nacimiento.

Para hacer referencia a un campo en concreto de un registro, se escribe, en primer lugar, el nombre del registro seguido de un punto y del nombre del campo.

writeln (FechaDeHoy. DiaMes + 1);

Como el campo DiaMes es de tipo entero, FechaDeHoy.DiaMes se puede utilizar exactamente igual que cualquier variable entera. Por tanto, la parte de instrucciones del programa cuyos datos hemos descrito antes podría ser:

```
begin
  FechaDeHoy.DiaMes:= 10;
  FechaDeHoy.Mes := Junio;
  FechaDeHoy.Anyo := 1987
end.
```



La estructura WITH

Está claro que resulta muy incómodo tener que escribir el nombre del registro cada vez que se utiliza un campo, pero es inevitable para no confundir los campos de dos registros del mismo tipo. Si en una parte del programa sólo se hiciera referencia a un registro en concreto, sería bueno poder decirle al compilador algo como: «bueno, en toda esta zona sólo voy a trabajar con el registro Tal, por lo que discúlpame de poner su nombre cada vez». Esto se puede hacer de la siguiente manera:

Se escriben, en primer lugar, las palabras reservadas WITH y DO con el nombre del registro entre medias, y detrás la instrucción en la que se va a omitir su nombre. Si fueran varias, se enmarcan por delante y detrás con las palabras reservadas BEGIN y END (dicho de otra manera: la instrucción puede ser estructurada y, por tanto, podemos poner una secuencia):

```
with FechaDeHoy do
  begin
    DiaMes:= 10;
    Mes := Junio;
    Anyo := 1987
  end;
```

La traducción del inglés sería:

«Con FechaDeHoy haz...»

En la mayoría de los compiladores la estructura WITH no sólo hace los programas más cortos y claros, sino que, además, permite hacerlos más rápidos.

El campo de un registro puede ser de cualquier tipo, incluyendo, por supuesto, otros tipos de registro. Vamos a ver cómo podría empezar un programa que manejara las fichas de la central lechera:

```
program Lechero;
(*----------------------------------------------------------------*)
(*                   DESCRIPCION DE DATOS                        *)
(*----------------------------------------------------------------*)
type
 Mes_t = (Enero,Febrero,Marzo,Abril,Mayo,Junio,Julio,Agosto,
          Septiembre,Octubre,Noviembre,Diciembre);

 Fecha_t = record
             DiaMes,Anyo : integer;
             Mes         : Mes_t
           end;

 Provincia_t = array [1..11] of char;

 Ficha_t = record
             Lote      : integer;
             FechaLote: Fecha_t;
             Destino   : Provincia_t
           end;

var
 LoteDeHoy: Ficha_t;

(*----------------------------------------------------------------*)
begin
...............

 with LoteDeHoy do   (* "Con LoteDeHoy haz:" *)
   begin
     Lote := 1234;

     (* FechaLote es un registro con tres campos: *)
     FechaLote.DiaMes  := 10;
     FechaLote.Mes     := Junio;
     FechaLote.Anyo    := 1987;

     Destino := 'Guadalajara'
   end;
...............

end.
```

Si no se utilizase WITH, habría que poner LoteDeHoy.FechaLote, para referirse a la fecha, y como ésta a su vez es un registro, habría que escribir LoteDeHoy.FechaLote.Mes para el mes en concreto. Sin embargo, se podría poner otro WITH para FechaLote:

```
with LoteDeHoy do
   begin
      Lote := 1234;

      with FechaLote do
         begin
            DiaMes := 10;
            Mes := Junio;
            Anyo := 1987
         end;

      Destino := 'Guadalajara'
   end;
```

o, de manera más clara:

```
with LoteDeHoy do
   with FechaLote do
      begin
         Lote := 1234;
         DiaMes := 10;
         Mes := Junio;
         Anyo := 1987;
         Destino:= 'Guadalajara'
      end;
```

Cada bloque WITH es en sí mismo una instrucción estructurada; tras «with LoteDeHoy do...», sólo hay, por tanto, una instrucción (otro WITH), por lo que no se utiliza el par BEGIN/END.

Se pueden poner unos bloques WITH dentro de otros siempre que no haya coincidencias entre nombres de campos de los diferentes registros, pues entonces el compilador no sabría a qué atenerse (aunque algunos lo toleran y suponen, por ejemplo, que en esos casos el campo es el del registro del primer WITH); la mayoría de los compiladores tienen un cierto límite para el número de bloques WITH que se pueden anidar unos dentro de otros.

Para simplificar estos casos de bloques WITH anidados, toda estructura del tipo:

```
with R1 do
   with R2 do
      .....
         with Rn do TalCosa;
```

se puede expresar, opcionalmente, poniendo:

```
with R1, R2,...Rn do TalCosa;
```

por ello, podríamos escribir:

```
 with LoteDeHoy, FechaLote do
   begin
      Lote := 1234;
      DiaMes := 10;
      Mes := Junio;
      Anyo := 1987;
      Destino:= 'Guadalajara'
    end;
```

Cuando el PASCAL encuentra una estructura WITH, toma nota de la porción de memoria en que se encuentra el registro, y es esa anotación la que utiliza para los campos. Por ello, en una situación como:

```
I := 3;
with Tabla (I) do
   for I: =1 to 10 do
```

en que Tabla fuese un ARRAY OF FICHA-T, siempre se mostraría el número de lote de la ficha 3, que es la que aparecía junto a WITH al llegar ahí. Para presentar los diferentes lotes lo correcto sería:

```
for I:=1 to 10 do
   with Tabla (I) do
      writeln (Lote);
```

para que cada vez que se pase por WITH la ficha sea la adecuada.

Veamos el ejemplo siguiente:

```
var
  A : integer;
  B,C : record
           A: char;
           B: boolean
         end;
```

Esta definición de variables, aunque pueda parecer errónea al utilizarse identificadores iguales para cometidos diferentes, es aceptable, pues siempre se le puede indicar de manera clara al compilador a qué nos estamos refiriendo:

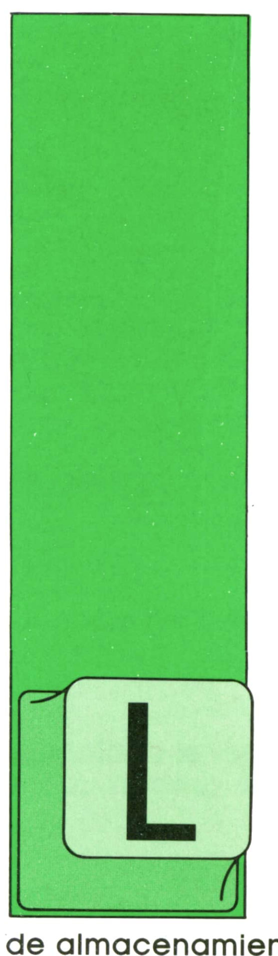
```
A := 3; (* el entero *)
B.A :='Z'; (* el carácter *)
B := C; (* el registro *)
B.B := true; (* el campo boolean *)
```

por otra parte, dentro de un bloque WITH, al encontrarse un identificador se busca primero entre los campos, por lo que no hay duda (ni posibilidad de utilizar las variables homónimas):

```
with B do
   begin
      A := 'Z'; (* el carácter *)
      B := true (* el campo boolean *)
   end;
```

Aunque en el PASCAL estándar dentro de ese bloque WITH se permite hacer referencia a «a» y «b» también como «b.a» y «b.b», con la mayoría de los compiladores existentes esto no es posible, debido a cómo funcionan internamente: al encontrarse el identificador «a», por ejemplo, se mira a ver si corresponde a alguno de los campos y como ése es el caso, no se mira más, por lo que al encontrarse el punto se produce un error.

OTROS LENGUAJES





Declaración de fichaje

LAS otras operaciones de entrada/salida que se han realizado hasta el momento han sido efectuadas a través del teclado y de la pantalla. Pero existen otros métodos de almacenamiento y grabación de datos, como son discos, cintas, listados de impresoras, etc.

Un fichero es un conjunto de datos, con determinada estructura lógica (registros). En una instalación puede haber un fichero de personal, con los datos de cada empleado: o un fichero de control de almacén, conteniendo el stock de cada artículo. Estos ficheros pueden encontrarse en distintos dispositivos periféricos.

La declaración de los ficheros que se van a utilizar en un programa se realiza en la ENVIRONMENT DIVISION. Una de las secciones de esta división es la INPUT-OUTPUT SECTION, que debe comenzar en el margen A y se divide en dos párrafos, siendo FILE-CONTROL el que más interesa en este apartado. Dicho párrafo debe comenzar en el margen A, mientras que las cláusulas que contiene lo hacen en el margen B.

Por cada fichero que se utilice en el programa debe emplearse una cláusula SELECT.

```
SELECT nombre-fichero ASSIGN TO periférico
                      ORGANIZATION IS LINE SEQUENTIAL
```

Hay que aclarar que esta cláusula es mucho más compleja de lo que se presenta aquí, pero sólo se explicarán los ficheros secuenciales (tienen los registros grabados uno a continuación de otro).

Es necesario, de todas formas, antes de compilar un programa, pedir información acerca de la SELECT del ordenador con el que se va a trabajar.

En el campo nombre-fichero se pondrá un nombre creado por el programador para distinguir ese archivo, se denomina nombre lógico.

Periférico se sustituye por el nombre con el que el compilador conoce al dispositivo en que se encuentra almacenado el fichero.

```
ENVIRONMENT DIVISION.
  INPUT-OUTPUT SECTION.
    FILE-CONTROL.
        SELECT PERSONAL ASSIGN TO DISK
                     ORGANIZATION IS LINE SEQUENTIAL.

        SELECT IMPRESO ASSIGN TO PRINTER.
```

El formato y estructura de los ficheros seleccionados se describe en la DATA DIVISION, es una sección especial conocida como FILE SECTION.

Comienza en el margen A y su formato es:

```
FD nombre-fichero                  STANDARD
LABEL RECORDS
                                   OMITTED
              VALUE OF FILE-ID «nombre físico»
```

En nombre-fichero debe ponerse el mismo nombre que se indicó en la SELECT.

Para ficheros asignados a discos se debe emplear la forma LABEL RECORD STANDARD. Además, deben llevar la opción VALUE, que sirve para relacionar el nombre del fichero en el disco y en el programa.

Si el fichero es una impresora, basta con poner LABEL RECORDS OMITTED.

A continuación debe detallarse la estructura del registro, siguiendo las reglas ya explicadas de los números de nivel.

La DATA correspondiente a la ENVIRONMENT anterior puede ser:

```
DATA DIVISION
 FILE SECTION
  FD PERSONAL LABEL RECORDS STANDARD
              VALUE OF FILE-ID "PERSO.DAT".

  01 REG-PERSONAL.
     05 NOM-APE      PIC X(35).
     05 DIRECCION    PIC X(25).
     05 TELEFONO     PIC X(11).

  FD IMPRESO LABEL RECORDS OMITTED.
  01 LINEA                  PIC X(80).
```

Instrucciones sobre ficheros

La primera instrucción a realizar con un fichero es abrirlo.

```
OPEN INPUT nombre-fichero
     OUTPUT nombre-fichero
     I-0    nombre-fichero.
```

El modo INPUT se utiliza con ficheros que sólo van a ser leídos. OUTPUT se emplea para ficheros que van a ser creados en el programa, mientras que I-0 se usará para los ficheros de los que se va a leer y en los que se va a escribir.

Todo fichero que se abre debe cerrarse:

```
CLOSE nombre-fichero
```

Para grabar en un fichero en disco se usa un WRITE:

```
WRITE nombre-registro.
```

Nombre-registro debe ser el declarado en la FD. Los registros se graban uno a continuación de otro.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. EJ-PERSONAL.

ENVIRONMENT DIVISION.

 INPUT-OUTPUT SECTION.

  FILE-CONTROL.

       SELECT PERSONAL ASSIGN TO DISK
                       ORGANIZATION IS LINE SEQUENTIAL.

DATA DIVISION.

 FILE SECTION.

  FD PERSONAL LABEL RECORD STANDARD
              VALUE OF FILE-ID "PERSO.DAT".

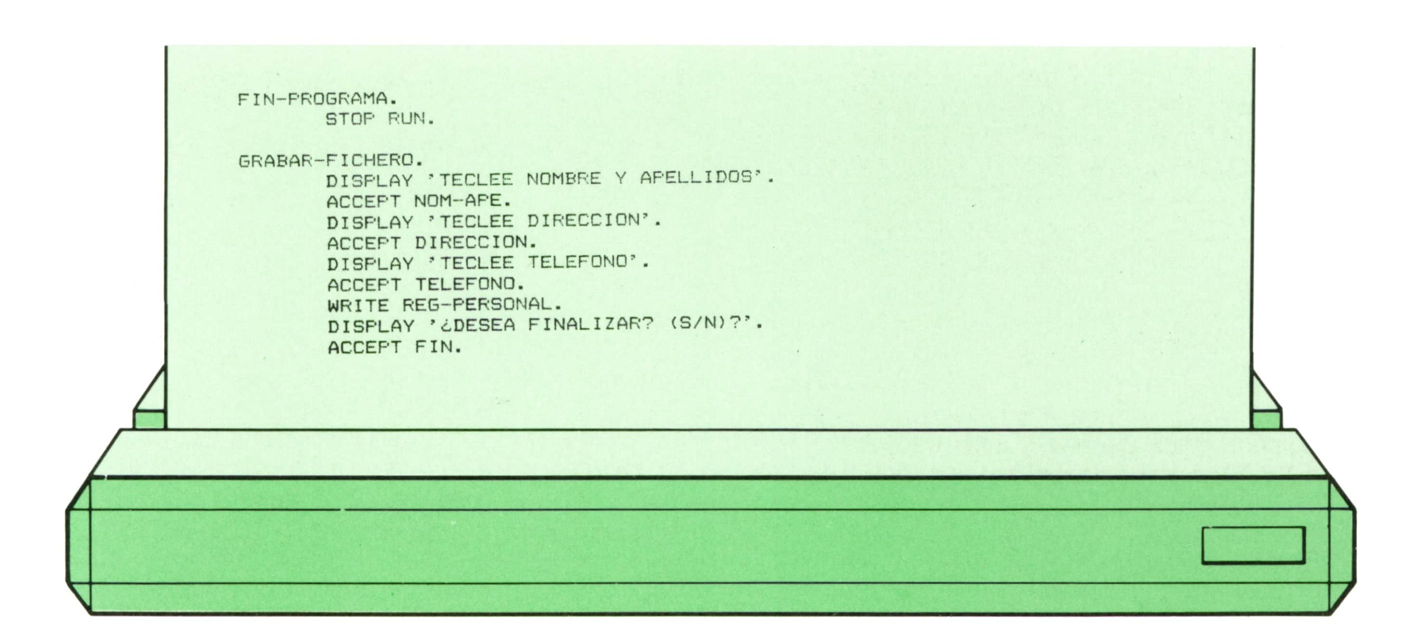
  01 REG-PERSONAL.
     05 NOM-APE          PIC X(35).
     05 DIRECCION        PIC X(25).
     05 TELEFONO         PIC X(11).

 WORKING-STORAGE SECTION.

  01 FIN                 PIC X.

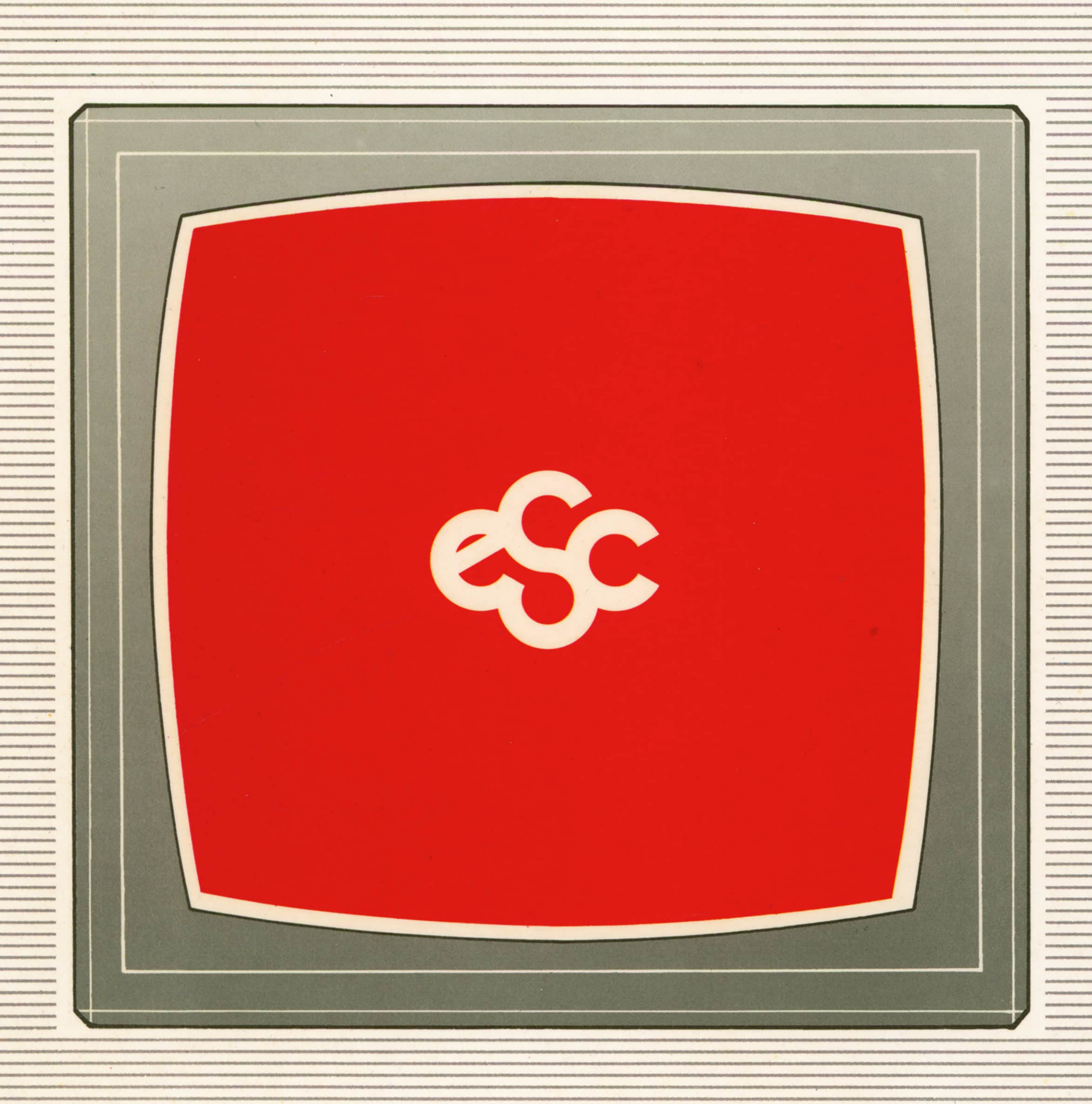

PROCEDURE DIVISION.


INICIO.
       OPEN OUTPUT PERSONAL.
       DISPLAY '¿DESEA FINALIZAR? (S/N)?'.
       ACCEPT FIN.
       PERFORM GRABAR-FICHERO UNTIL FIN = 'S'.
       DISPLAY 'FICHERO CREADO'.
       CLOSE PERSONAL.
```



```
FIN-PROGRAMA.
        STOP RUN.

GRABAR-FICHERO.
        DISPLAY 'TECLEE NOMBRE Y APELLIDOS'.
        ACCEPT NOM-APE.
        DISPLAY 'TECLEE DIRECCION'.
        ACCEPT DIRECCION.
        DISPLAY 'TECLEE TELEFONO'.
        ACCEPT TELEFONO.
        WRITE REG-PERSONAL.
        DISPLAY '¿DESEA FINALIZAR? (S/N)?'.
        ACCEPT FIN.
```



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼